FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

## Master's Thesis

# Muscle interaction in the context of muscle deformation modelling by a Position Based Dynamics method

Bc. Ondřej Havlíček

**Thesis advisor**
Doc. Ing. Josef Kohout, Ph.D.

PILSEN, CZECH REPUBLIC 2024

**Citation in the bibliography/reference list:**
HAVLÍČEK, Ondřej. *Muscle interaction in the context of muscle deformation modelling by a Position Based Dynamics method*. Pilsen, Czech Republic, 2024. Master's Thesis. University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering. Thesis advisor Doc. Ing. Josef Kohout, Ph.D.

University of West Bohemia
Faculty of Applied Sciences
Academic year: 2023/2024

# ASSIGNMENT OF DIPLOMA THESIS

(project, art work, art performance)

| | |
|---|---|
| Name and surname: | **Bc. Ondřej HAVLÍČEK** |
| Personal number: | **A22N0002P** |
| Study programme: | **N0613A140037** |
| Specialization: | |
| Work topic: | **Muscle interaction in the context of muscle deformation modelling by a Position Based Dynamics method** |
| Work topic in English: | **Muscle interaction in the context of muscle deformation modelling by a Position Based Dynamics method** |
| Assigning department: | **Department of Computer Science and Engineering** |

## Theses guidelines

1. Describe the problem of intermuscular interaction.
2. Describe current approaches to computer modelling of the interaction of deformable bodies.
3. Design and implement an approach to avoid penetration of muscles.
4. Extend the simulator using different PBD parameters for different muscles, representing mutual interaction during various movements.
5. Verify the solution on the available data of muscles in the pelvic area during flexion, extension, abduction and internal rotation of the leg.
6. Critically evaluate the results.

| | |
|---|---|
| Length of diploma thesis: | **50 pages recommended** |
| Extent of graphics content: | **not specified** |
| Form processing of diploma thesis: | **printed/electronic** |
| Language of elaboration: | **English** |

Recommended resources:


- Kohout, J. & Červenka, M. Muscle Deformation Using Position Based Dynamics. Biomedical Engineering Systems and Technologies, Springer International Publishing, 2021, 486-509. DOI:10.1007/978-3-030-72379-8_24


| | |
|---|---|
| Supervisors of diploma thesis: | **Doc. Ing. Josef Kohout, Ph.D.** |
| Reviewer of diploma thesis: | **Mgr. Martin Maňák, Ph.D.** |
| Date of assignment of diploma thesis: | **September 8, 2023** |
| Submission deadline of diploma thesis: | **May 16, 2024** |

L.S.

**Doc. Ing. Miloš Železný, Ph.D.**
dean

**Doc. Ing. Přemysl Brada, MSc., Ph.D.**
Head of Department

V Plzni October 11, 2023

# Declaration

I hereby declare that this Master's Thesis is completely my own work and that I used only the cited sources, literature, and other resources. This thesis has not been used to obtain another or the same academic degree.

I acknowledge that my thesis is subject to the rights and obligations arising from Act No. 121/2000 Coll., the Copyright Act as amended, in particular the fact that the University of West Bohemia has the right to conclude a licence agreement for the use of this thesis as a school work pursuant to Section 60(1) of the Copyright Act.

Pilsen, on 14 February 2024

. . . . . .

Ondřej Havlíček

# Abstract

This thesis focuses on modelling intermuscular interactions in the context of muscle deformation using the Position Based Dynamics (PBD) method. The deformation method was extended to improve its consistency during simulation. Passive intermuscular interaction was introduced and implemented through a general deformable collision handling system. Additionally, a system for active muscle interaction was developed for the model, enabling physiologically accurate, motion-based, and synchronous muscle contractions during arbitrary movements. The results were rigorously verified against a similar method and evaluated for physiological accuracy, preservation of shape and volume, and muscle collision avoidance. The physiologically advanced model seems to hold a real-world application potential.



# Abstrakt

Práce je zaměřena na modelování interakce mezi svaly v kontextu deformace svalů metodou Position Based Dynamics (PBD). Metoda deformace byla rozšířena za účelem zlepšení konzistence během simulace. Koncept pasivní mezisvalové interakce byl představen a implementován pomocí obecného řešení deformovatelných kolizí. Kromě toho byl pro model vyvinut systém pro aktivní interakci svalů, který umožňuje fyziologicky přesné, na pohybu založené a synchronní kontrakce svalů během libovolných pohybů. Výsledky byly důkladně ověřeny proti podobné metodě a hodnoceny z hlediska fyziologické přesnosti, zachování tvaru a objemu a vyhýbání se kolizím svalů. Zdá se, že tento fyziologicky pokročilý model skýtá potenciál pro aplikaci v reálném prostředí.



## Keywords

Extended Position-Based Dynamics • Musculoskeletal system • Deformation • Simulation • Deformable bodies • Muscle contraction modelling • Deformable collision detection • Collision detection and resolution • Muscular synergy • C++

# Acknowledgement

I would like to acknowledge and express my gratitude to my supervisor, Doc. Ing. Josef Kohout, Ph.D., for his long-term guidance, which has enabled me to navigate difficult challenges, such as writing this thesis. His inspiration not only motivates me but also encourages other students to pursue further research and fulfil their ambitions, cultivating the academic spirit.

I would also like to thank my family and close ones for being patient with my absence in daily life and always supporting me even in the most difficult situations.

# Contents

# Introduction 1

"In my dreams I have my leg, I'm running… I never dream of myself without my leg", explains one post-amputation idiographic analysis [Roș+21] participant. One of the most prevalent skeletal diseases is osteoporosis in 34.3% US and Japanese women above 50 years of age [Wad+14] causing the brittleness of the bones, possibly leading to fracture and a cascade of serious health consequences, including amputation.

Helping medical professionals decide whether to perform invasive surgery on the musculoskeletal system and prevent such consequences can be achieved through a computerised musculoskeletal model. These models come in varying complexities, using straight segments to represent muscles [Hei+23], and bone-wrapped segments may be used for hip area [De +18] muscle modelling. Arguably, non-realistic muscle approximation (e.g. not taking volume preservation into account) may lead to misleading clinical decision assistance [KČ21]. Hence, the search for a more realistic computer musculoskeletal model continues.

One such musculoskeletal system based on a Position Based Dynamics method (PBD) ([KČ21]) exhibits unnatural bending of *musculus iliacus* being passively dragged in a rag-doll manner during hip flexion. This visual representation is often the indirect measurement of the model quality due to the lack of systematic modelling evaluation methods [DT18]. The unrealistic *iliacus* bending motivates a model better reflective of the underlying muscle physiology, taking muscle contraction, relaxation and collaborative interplay into account.

Therefore, the main focus of this thesis is to extend the current PBD-based simulator to support muscular interactions during various movements, where the different muscles usually act in different roles, including contraction, relaxation, fixation and so on. All this while keeping up with the model requirement to execute simulations in real-time, it seems sensible to implement the muscle interplay through various PBD particle parameters of the current model. During these muscle interaction simulations, the muscles should also not intersect each other.

# The physiology of the skeletal muscle — 2

To develop appropriate muscular cooperative movement strategies, it is first crucial to understand muscle structure and physiology (their function). To know how the muscles should interact, how they act individually should first be explained. The muscles being modelled are skeletal muscles mostly made of striated muscle fibres. These fibres are, as opposed to the smooth muscle fibres, voluntarily controllable [Gas+24] by the Central Nervous System (CNS) [BC13], specifically through the cranial and spinal nerves under the control of the cerebral cortex [RMT16].

## 2.1 Striated muscle structure and function

The muscle (illustrated in Figure 2.1), protected from friction by the epimysium, comprises fascicles grouping muscle fibres that run in parallel. The fascicles, covered by another protective layer of connective tissue called perimysium, are made up of several tens of muscle fibres. The muscle fibre is a cell containing many nuclei (also called syncytium), anatomically and functionally separated from other muscle fibres by the sarcolemma membrane. Among other organelles like mitochondria, the fibres contain thousands of rod-like myofibrils running in parallel. In turn, the myofibrils are composed of actin (thin) and myosin (thick) myofilaments. These filaments are interlaced into basic functional units called sarcomeres, describing the myofibril segment between two surfacing $Z$-shaped patterns (Z-lines) that emerge from actin filaments layout while in between on the surface, there emerges an $M$-shaped pattern out of the myosin filaments connectivity (M-line). Myofibril is also covered by the sarcoplasmic reticulum which regulates the levels of calcium, a crucial element for contraction. The sarcoplasmic reticulum contains encircling invaginations called T-tubules propagating contraction signal further the myofibril usually situated at the place of the Z-line which is also called the Z disc.

The thick filaments contain the myosin proteins that give rise to pairs of heads, where actin-binding and ATP-binding sites exist. The thin filaments then contain the actin, tropomyosin and troponin proteins [Gas+24]. The actin proteins have myosin-

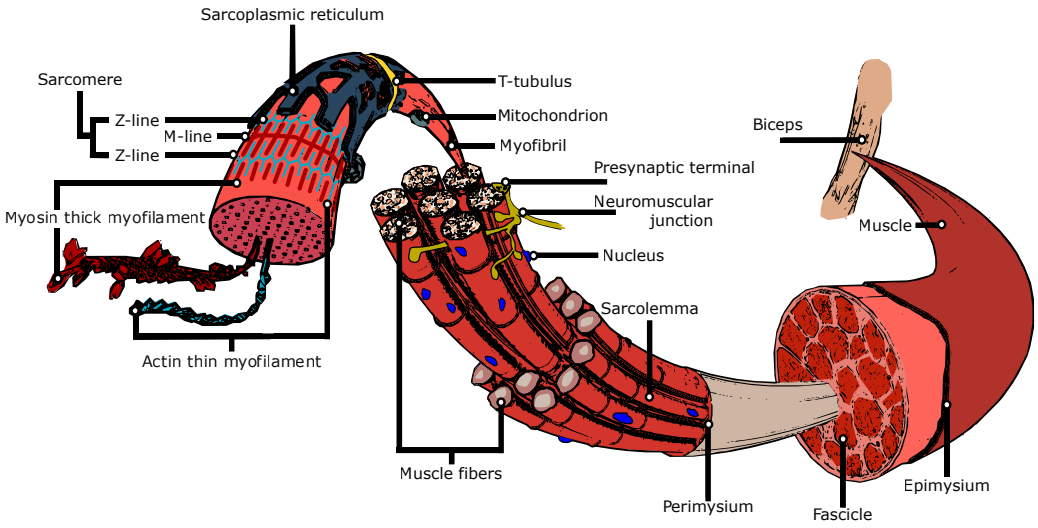


Figure 2.1: The structure of skeletal muscle, source: [Ólaı17] (vectorised and modified)

binding sites, which are, in a relaxed state, blocked by the regulatory tropomyosin proteins. Troponin serves as a kind of transmitter regulatory protein to facilitate the contraction.

These filaments, according to the sliding filament theory described by Hugh Huxley and Jean Hanson in 1954, produce electrochemically governed contraction by trying to slide against each other through myosin pulling onto actin via so-called cross-bridges [Gas+24], causing the sarcomeres as well as the myofibrils and therefore the whole muscles to shorten while maintaining the same muscular tension (isotonic contraction). Thanks to the elasticity of the muscle due to the titin protein presence, the contraction is also possible without the length change, where instead, the muscle tension grows (isometric contraction), which is also the basis for the isotonic contraction [RMT16]. Further explanation follows in Section 2.1.4.

According to [Gas+24], the contraction is the muscle's primary function. For this important locomotion process to begin, though, a signal from the CNS must first arrive.

### 2.1.1 Neural signalling

The signal (also called the action potential or cell membrane depolarisation) to contract a muscle, originating either somewhere in the cerebral cortex or just as a part of some spinal cord reflex, is carried through myelinated efferent (motor) nerve cells (motoneurons).

The motoneurons can be classified into central (descendent) motoneurons, which carry the action potential from the cerebral cortex to the spinal cord, and the peripheral motoneurons, finally joining at the skeletal muscle through the neuromus-

cular junction (Figure 2.1). This one neuron, as shown in the figure, usually innerves a group of striated muscle fibres called the motor unit. The number of fibres in the unit influences the speed and precision of the movement activity [RMT16]. For example, among extraocular muscles, one motoneuron governs only 6 muscle fibres, while on the other hand, musculus *biceps brachii* contains one motoneuron governing approximately 750 fibres [RMT16].

## 2.1.2 Neuromuscular junction

When the action potential reaches the interface between the motoneuron and the muscle, depicted in Figure 2.2, the neuromuscular junction at the unmyelinated, widened neuron terminal part called the terminal bouton, is now described as the presynaptic action potential. It depolarises the axon terminal, causing voltage-gated [OMB24] calcium $Ca^{2+}$ channels on the presynaptic motoneuron membrane to open, letting it inside the cell via concentration gradient, increasing $Ca^{2+}$ ion concentration inside the bouton which begins the process of exocytosis of the acetylcholine (ACh) mediator inside the synaptic vesicles [OMB24].

During the exocytosis, so-called SNARE (Soluble N-Ethylmaleimid-sensitive fusion protein Attachment protein REcepetor) proteins make the ACh vesicles fuse with the membrane [OMB24], realising ACh into the synaptic cleft, where it travels via diffusion towards the cholinergic nicotinic ACh ligand-gated ion channel receptors on the myofibril sarcolemma (postsynaptic membrane) binding sites called the junctional folds.

Binding two ACh onto the ligand-gated receptor opens the channel and allows sodium ($Na^{+}$) ions into the myofibril depolarising the whole membrane. Depolarisation of the postsynaptic membrane past a certain threshold (also called the end-plate potential) creates an action potential ($*AP$) anywhere on the sarcolemma except for

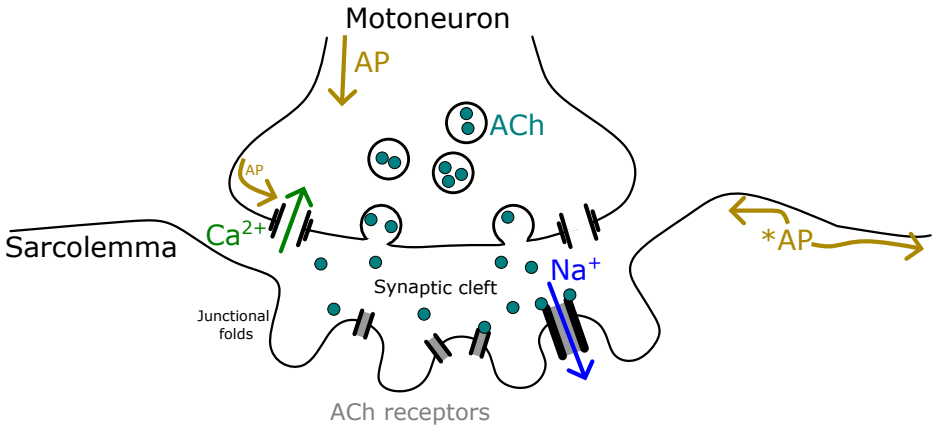


Figure 2.2: Simplified illustration of sarcolemma excitation

this postsynaptic membrane area which starts to spread in every direction.

### 2.1.3 Excitation-Contraction coupling

Once the action potential reaches the so-called T-tubule, an invagination in the sarcoplasmic reticulum on the sarcolemma, it travels down to the so-called triad, the green union depicted in Figure 2.3. There, a process called excitation-contraction coupling begins. The first receptor to meet the action potential is the **Di**hydro**p**yridine voltage-gaited calcium channel, where it causes a conformational change [Gas+24]. This change in the RHP receptor proteins mechanically causes nearby **Ry**anondine **r**eceptors to release large amounts of calcium from the calcium storage (dark-blue area in Figure 2.3) within the sarcoplasmic reticulum. Some of the released calcium ions make it to the Troponin C protein on the thin filaments, which causes a conformal change in the Tropomyosin proteins, uncovering the binding sites on Actin proteins for Myosin heads. Once the sites are uncovered, the myosin heads use the ATP and phosphates to form the so-called cross-bridges. To form this connection, the myosin heads actively extend forward and outwards from the thick filament. Upon forming the cross-bridge, the myosin head pulls back in a swim-like motion,

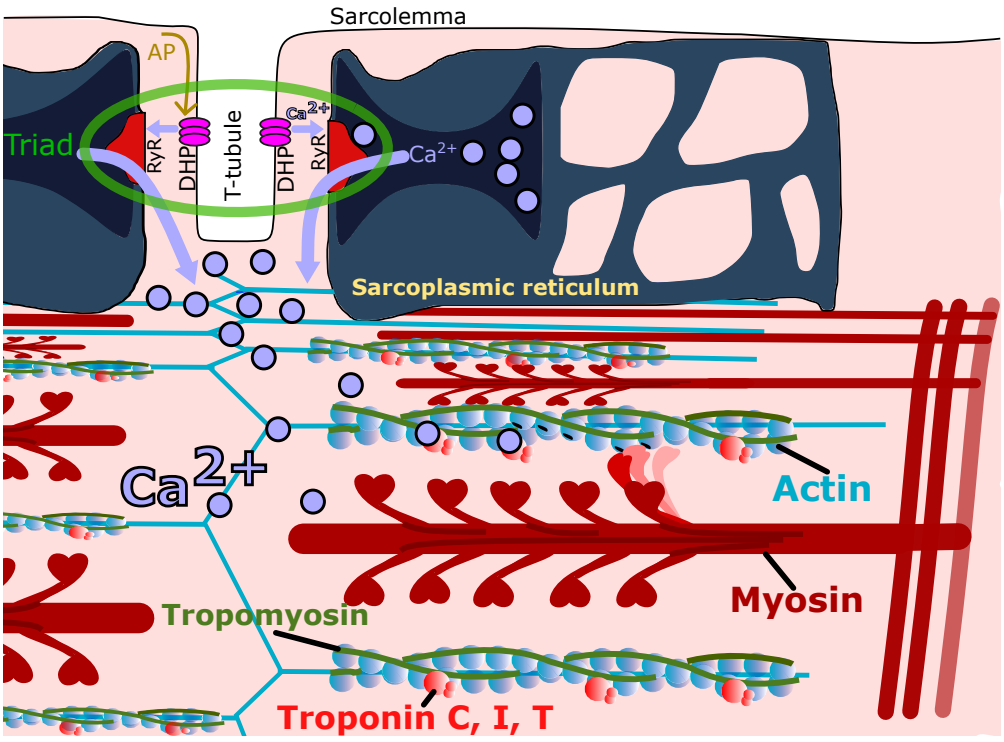


Figure 2.3: T-tubule Excitation-Contraction coupling scheme (`www.wikipedia.org/wiki/Myofilament` graphic used)

which is then repeated as long as the calcium is present [Gas+24], causing the muscular tension to rise and finally the contraction to occur.

The relaxation begins when the calcium levels drop, blocking the actin-myosin binding sites, and making the filaments slide back to the original position, also thanks to other present elastic proteins. Although relaxation might be an active process from the point of view of cortical activation in that one can decide to relax a muscle, in the muscle itself, it probably just is an act of starving the contraction of calcium. The tension inside the muscle fibres generates different types of contraction.

### 2.1.4 Types of contraction

The type of contraction is discriminated by the states of the muscle tension and the corresponding fibre length, generally into four types.

Rising tension inside of the muscle while it maintains constant overall length is called the **isometric** contraction. During this type of contraction, there is no motion of the limbs or joints, while the volume of the muscle most probably does not change, hence the tension is mostly expressed as the raising of the internal pressure, pulling on the tendons, making the muscle thicker in shape around its belly and thinner in shape near the tendons. When the tension is constant and the fibre lengths change, then **isotonic** contraction emerges (where the muscle volume does not change). **Isotonic concentric** contraction occurs when the muscle tension becomes greater than the external force, shortening the muscle fibres, and causing tension at the tendons to heighten which in turn moves the bones on the muscle insertion and stabilises the muscle at its origin. **Isotonic eccentric** contraction serves to prevent joint damage by acting as a brake to the concentric contraction [Gas+24].

The direction of the contraction is dictated by the direction of individual muscle fibres. Indeed, inside a *fascicle* (as illustrated in Figure 2.1) the fibres are always running in parallel. But the fascicles themselves do not always have to be arranged in a parallel manner. The fascicles may be arranged in a **parallel**, **convergent**, **pennate**, **fusiform**, **spiral** or **circular** structure. On top of that, the tendons intertwining the fascicles also emerge in numerous arrangements, further discriminating the structures and function of the muscles.

But to perform a complex movement, e.g. hip flexion, a spectacular synchronous orchestra of such different contractions and relaxations usually occurs.

## 2.2 Roles of muscles in movements

A main muscle is always responsible for any movement, the so-called **prime mover** [Bet+13], or the **agonist** muscle. Each movement caused by an agonist has a so-

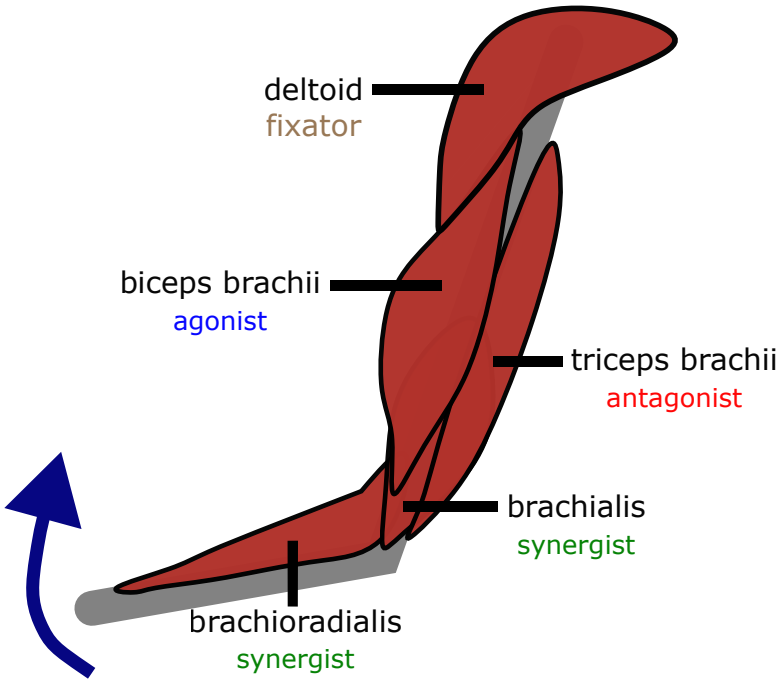


Figure 2.4: Roles of muscles during elbow flexion

called **antagonist** muscle, usually located on the opposite side of the bone. For example, if the muscle *biceps brachii* contracts as a prime mover, its antagonist, *triceps brachii* must relax. On the other hand, if one extends the elbow, *triceps brachii* now acts as a prime mover to which *biceps brachii* opposes by relaxation (from the former contraction during elbow flexion). Not only does the antagonist relax, but can also help moderate body and limb positions in space and during rapid movements [Bet+13].

The rest of the muscles involved in the movement are called the **synergists**. The synergist muscles may act in various ways to assist the prime mover. They contribute to the contraction and also help prevent unwanted movements [Bet+13]. A type of synergist muscle is a **fixator**, stabilising the agonist's origin (a non-moving bone area the agonist holds onto). A *brachialis* muscle together with the *brachioradialis* muscle are usually the synergist to the prime mover during elbow flexion (note that the second muscle is positioned distally from the *biceps brachii* insertion, across the elbow joint). The *deltoid* would play the role of a fixator in this situation. The muscles and their roles during elbow flexion can be seen in the illustration 2.4.

### 2.2.1 Muscular synergies

Not only do the motoneurons affect many motor units simultaneously, but it is probable that also these motoneurons are likely not controlled individually, but

rather in groups called the **muscle synergies** [BC13]. This does not oppose the existence of other movement sources such as various reflexes. The estimated number of synergies acting during for example the running or walking movement is between four and five. At the same time, these synergies are mostly subject-specific and can be trained, and changed throughout life [BC13]. This may be one of the explanations of how the CNS can orchestrate such complex movements as e.g. throwing the discus in such an efficient manner or even how exactly the agonist and antagonist function simultaneously.

## 2.3 Orchestrating complex movement

Consider the hip area. Even just keeping the hip joint stable (that is to say, keeping the *femoral* head in contact with the hip joint cavity called the *acetabulum* [KOA19]), periarticular muscles (around the joint) such as the *piriformis*, *obturator externus*, *gluteus maximus*, *medius*, *minimus*, and more, are co-activated in a group called the muscular *fasteners* of the hip joint [KOA19]. Meanwhile, longitudinal muscles (lengthwise along the body) usually push the *femoral* head upwards. Many other muscles try to keep the *femoral* head well oriented to the *acetabulum* as this is an important factor of stabilisation [KOA19].

From this standing position, six basic movements of the *femoral* bone within the hip joint are usually described in the joint with three degrees of freedom (movement in each axis split into two). Respective axes are the **transverse axis** (up and down), the **sagital axis** (left and right), and the **vertical axis** (roll by the right or left hand). The movements are described in the 2.1 table with respective ranges of motion (ROM) in an upright position with an extended knee for an average person [KOA19].

Table 2.1: Movements of the *femur* within the Hip Joint

| Movement | Direction | ROM [degrees] | Axis |
|---|---|---|---|
| Flexion | up | 0-90 | transverse |
| Extension | back | 0-20 | transverse |
| Abduction | to the side | 0-30 | sagital |
| Adduction | to the centre | 0-30 | sagital |
| Internal (medial) Rotation | inwards | 0-60* | vertical |
| External (lateral) Rotation | away from the body | 0-30* | vertical |

* when lying on the stomach with a flexed knee

These ranges of motion are otherwise highly dependent on other factors, such as knee flexion, *pelvis* rotation, the subject's level of athleticism, injuries, or activity/-passivity of the movement (passive movement occurs when the rest of the body puts the joint in that position). For example, hip flexion with the knee bent can usually

reach up to **120°** thanks to the relaxation of the *hamstrings*, even more, if the flexion is passive, and up to **145°** if the arms are used to press the knee towards the thorax. Both hips can also be completely flexed in a slouching sitting position on the ground with the knees tucked to the chest [KOA19].

With hip abduction, some trained ballerinas can reach active abduction of up to **130°** [KOA19]. Thanks to forward pelvic tilting, trained individuals can also reach the extension of the hip up to **90°** (while doing the splits). These movements are achieved with various muscles acting at various times, often changing the roles they play in the movement as it progresses.

### 2.3.1 Hip flexion

The most powerful actors during the hip flexion are the *iliacus*, the *psoas major* and the *psoas minor*, which together are called the *iliopsoas*. This muscle also produces lateral rotation to a degree. The *sartorius* muscle is also a hip flexor, while the *rectus femoris* contributes to the flexion dependent on the knee flexion, to which its contribution to the hip flexion is proportional [KOA19]. Other hip flexion synergist muscles are the *tensor fasciae latae*, the *pectineus*, the *adductor longus* and the anterior (the front) fibres of the *glutei minimus* and *medius*.

These muscles are further functionally classified into two groups:

1. Flexion - abduction - medial rotation group
    - *anterior* fibres of the *glutei minimus* and *medius*
2. Flexion - adduction - lateral rotation group
    - the *iliopsoas*, the *pectineus* and the *adductor longus*

which together, in a coordinated fashion, through muscle synergies, have the capability of producing so-called **pure flexion** (e.g. during walking) provided they work as a "balanced set of synergists and antagonists" [KOA19].

### 2.3.2 Hip extension

The muscles producing hip extensions are classified into two groups depending on their insertion areas:

1. Muscles inserted into the upper extremity of the *femur*
    - the *glutei maximus*, *minimus* and *medius*
2. Muscles inserted into the vicinity of the knee

- the *hamstrings*, the *biceps femoris*, the *semitendinosus* and the *semimebranosus*

The extension of the knee accelerates the efficiency of the second group's function during the hip extension, uncovering the opposing interaction of the *hamstrings* and the *rectus femoris* [KOA19].

Out of all of these muscles, two groups of secondary functions can be defined. Those which also produce abduction and those which also produce adduction. These groups roughly correspond to the discrimination by insertions except for the *semitendinosus*, the *semimembranosus* and the *biceps femoris*, where with additional adductor muscles, the two groups are capable of producing the so-called **pure extension**, where, again, the groups must contract in well-balanced synergist-antagonist fashion [KOA19].

An important note is that during the gait on a flat surface, the *gluteus maximus* **does not actively perform the extension**. Instead, the *hamstrings* take over. On the other hand, the most powerful muscle extends the hip while walking up a slope, and also during running or jumping [KOA19]. Other roles of the extensor muscles include the stabilisation of the *pelvis* during tilting.

### 2.3.3 Hip abduction

To abduct the hip, the *gluteus medius* is the primary mover since it is almost perpendicular to the lever arm of the *femur* head. The synergists here are the *gluteus minimus*, the *tensor fasciae latae* and the highest fibres of the *gluteus maximus*, along with the *piriformis* [KOA19].

Once again, classified into two groups based on their secondary function, these muscles, given well-synergized interaction, produce **pure abduction**. The classification is as follows [KOA19]:

1. Muscles anterior to the coronal plane (plane slicing the body in half while looking at it from the side at the centre of the joint), also producing flexion
    - the *tensor fasciae latae*, anterior fibres of the *glutei minimus* and *medius*
2. Muscles behind the coronal plane, also producing extension
    - posterior (further back) fibres of the *glutei minimus* and *medius*, the *semitendinosus* and the *semimebranosus*

The pure abduction can also be facilitated by a synergized contraction of the so-called **deltoid of the hip** made of two muscle bellies, which are the *tensor fasciae latae* and the superficial fibres of the *gluteus maximus*. These both insert into the border of the *iliotibial tract* in the thigh [KOA19].

The *gluteus medius* firstly initializes the movement while mostly staying in the role of stabilizing the joint, progressively acting more and more as an abductor and less as a stabilizer with a higher degree of abduction, while being the strongest at approximately **35°** abduction [KOA19]. Similar situations are quite normal to occur even to the most effective prime movers of various movements.

### 2.3.4 Hip adduction

Once again, the muscles of adduction are mostly governed by kinematic advantages. The prime example is the *adductor magnus*, which spans along the majority of the *femur* body, along with *adductor minimus*, *adductor longus* and *adductor brevis*, being able to pull on it across a wide spectre under beneficial angle [KOA19]. Other muscles contributing to the adduction are the *gracilis*, the *semimembranosus*, the *semitendinosus* and the *biceps femoris* (although these muscles are mainly used for hip and knee flexion), lower fibres of the *gluteus maximus*, the *quadratus femoris* and *pectineus* (which also take part in lateral rotation), and lastly, the *obturator internus* and *externus* also contribute to the movement as synergists [KOA19]. Of course, to achieve adduction, all these muscles must function in a coordinated fashion together with other muscles, which e.g. stabilize the joint.

### 2.3.5 Hip external rotation

Muscles whose primary function is to externally (laterally) rotate the hip are the *piriformis*, the *obturator internus*, and also the *obturator externus*, which in contrast to the others, truthfully wraps around the *femoral* neck. These muscles are called the *pelvitrochanteric* (around the hip) muscles. Adding to the collection are the muscles with external rotation as their secondary function, which include the *quadratus femoris*, the *pectineus* (which has already been described to also contribute to adduction and flexion), the posterior fibres of the *adductor magnus*, and partly also all *glutei* [KOA19].

Moreover, when the knee is extended and the *femur* rotates in a standing position, muscles like the *biceps femoris*, the *semitendinosus*, the *semimembranosus*, and partly the adductors also provide to the lateral rotation but also contribute to the internal (medial) rotation when the leg rotates inwards. The action of these muscles depends on their spatial relationship to the *femoral* vertical axis. When the muscles are anterior to it, they rotate medially, and while they are posterior to it, they rotate externally [KOA19].

### 2.3.6 Hip internal rotation

Muscles causing the internal (medial) rotation are running anterior to the *femoral* vertical axis usually producing about one-third of the force of the external rotators [KOA19]. The primary muscles are the *gluteus medius*, the *gluteus minimus* and the *tensor fasciae latae*.

The muscles *pectineus* and *obturator externus* lose their function as external rotators at the point of approximately **30-40°**, because of their new position directly under the joint centre, no longer being posterior to the *femur*. Instead, after full medial rotation (approximately **60°**), these muscles start to act as medial rotators, while the primary hip internal rotators now become external ones [KOA19].

### 2.3.7 Muscular action inversion

As described in the previous section, muscle roles in the movements and times of contraction may vary greatly due to various conditions. A muscle causing movement in one direction, usually as its secondary function, can change its role to cause a movement in the opposite direction (e.g. A flexor becomes an extensor).

This is called the inversion of muscular action [KOA19] due to the muscle fibres' direction and position relative to the joint axes and the surrounding joint orientations. For example, the *iliopsoas* can no longer flex the hip at the ballerina's maximum **120°**, since its tendon changes the directions of the fibres. The muscles are usually recruited successively during whole movements and may stop contributing to the movement at its fullest [KOA19].

## 2.4 Summary

It becomes obvious that the actions of the hip muscles can not be studied in detail without the knowledge of the surrounding conditions such as pelvic tilt, knee flexion level, the centre of gravity of the body, or the individual's training level.

On top of that, many muscles act differently under different movements. To generally capture how exactly the hip muscles should contract during an arbitrary position is a challenging task. That is why the functions of the muscles are often better explained through the detection of electrical responses (potentials) to stimuli using **electromyography** (EMG). On the surface, the measurement is non-invasive, but identifying the precise source of the electrical potentials can be problematic, where approaches like blind source separation can be utilised, similar to identifying sources of electrical potential on the surface of the brain captured through electroencephalography. To measure the muscle activity directly means to invasively insert the measuring electrodes into the muscle tissue, which is usually not a standard procedure for healthy humans.

Luckily, many biomechanical simulation methods exist to estimate muscular activity during movements, some of which are provided by e.g. the **OpenSim** (simtk.org/projects/opensim) software. These methods are usually well-documented in their limitations and validated against the EMG ground results.

Now that the muscular function should be at least partly explained together with some specific primary and secondary muscle groups during various movements of the hip joint, it is necessary to note the level of complexity the muscles embody. Be it the physiological mechanism of how the muscle's primary function, **contraction**, comes to exist, **possible structures** the muscle may be arranged in, **types of contractions**, and finally the muscular **interplay** and the **agonist-antagonist-synergist-fixator** relationships.

The musculoskeletal system can be modelled in different levels of detail depending on the aim of the research. To model the system, where the muscles are elastic, maintain their volumes, do not intersect, and can interact (synergistically contract) is the main challenge of this thesis.

# Modelling deformable bodies — 3

Advances in mechanical modelling, computer graphics, and many other related fields, have brought a broad array of methods to represent the muscular structure, which further more or less dictates its possible function. One of the main attributes of the muscles is that they are elastic and their geometries deform during movements.

But to model a deformable body (object) means to decide on the structure and the method to deform that structure over time under external (e.g. gravity) or internal (e.g. tension) forces appropriately. To intuitively imagine the bodies, this chapter will illustrate the objects and deformations in two or three-dimensional space, although usually, no such dogmatic constraint applies to the structures or methods themselves.

## 3.1 Object representation

When a real-world object is computerised (perhaps by a designer, an artist, or an imaging modality), it can be represented by a set of mathematical functions approximating the real object shape **continuously** or by a set of **discrete samples** (often in the form of a mesh or a point cloud). This representation is usually dependent on the method of model acquisition and the need for detail. Even though there are always discrepancies between the model and the real-world objects, the approximation usually suffices the goal (e.g. to visualise the object, interact with it, or perform an analysis of the object's behaviour, etc.).

In case the object is represented by discrete samples, one sample is called the **vertex** being associated with a position that can be expressed relative to a global Cartesian coordinate system origin along with the other vertices.

### 3.1.1 Mesh-less

If no other information is provided, the representation can be described as a **mesh-less** one. It can be useful in e.g. fluid modelling, and other models, where the con-

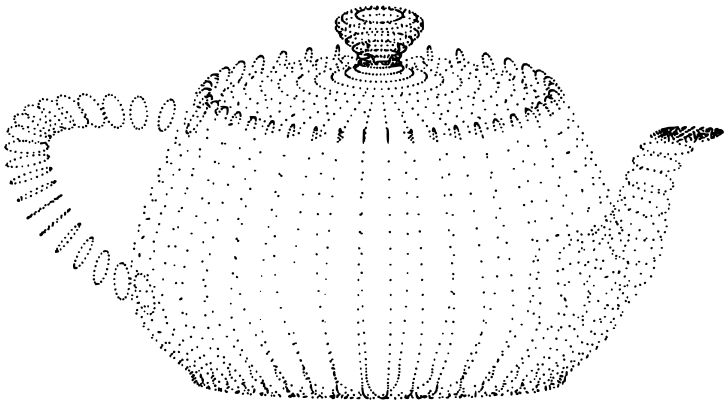

Figure 3.1: Mesh-less representation of the Stanford teapot

nectivity of samples is not important or can not be directly determined. An example can be seen in Figure 3.1.

One drawback may be that the information about the neighbourhood of one point is not so straightforward, and usually space partitioning methods, like **K-means** have to be used to obtain K-nearest neighbours and even then it is not obvious if these neighbours all lie on the same side of the underlying surface as the query point if that is a desirable piece of information.

A popular approach to acquire a mesh-less 3D representation is the **LiDAR** method (oceanservice.noaa.gov/facts/lidar.html) which measures sample depth by shooting a ray towards the surface and elapsing the time it takes for the ray to return to a receiver. From the set of vertices, generally non-uniform, an approximation of the surface can be reconstructed. The approximation error is, of course, dependent on the density of the samples.

### 3.1.2 Surface mesh

The additional connectivity (also topological or shape) information can be either explicitly modelled but also acquired through **surface reconstruction**, where a pipeline usually involves some type of segmentation of the mesh-less samples and subsequent surface extraction. It is often represented by a set of polygons (e.g. triangles or quads) approximating the original real-world surface shape. A surface triangular mesh is depicted in Figure 3.2.

The main advantage compared to the mesh-less representation is that the software structure defining the surface mesh usually provides a query to obtain the neighbourhood of a query vertex or polygon in a constant time and this relation between vertices is well-defined through the edges (sides of the polygons).

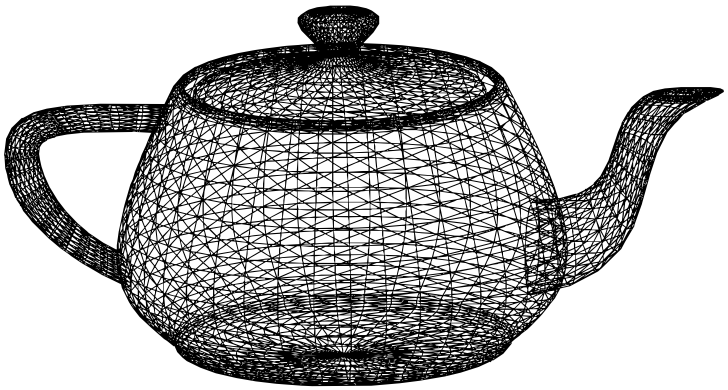

Figure 3.2: Surface mesh representation of the Stanford teapot

### 3.1.3 Volume mesh

If information about the internal contents of the object is also provided, then the mesh is volumetric. Imagine the object sliced into small cubes without changes in positions, then it could be called a *voxel* volume mesh, one of which is illustrated in the picture 3.3. Other standardly used primitives are the tetrahedra (as an extension of the triangular mesh). The tetrahedra can be obtained by slicing the cube through four of its corners, where one extra tetrahedra forms in the middle of it, producing **five** tetrahedra in total, or through slicing of the cube such that the resulting tetrahedra share the diagonal of the cube, while the other three points are formed on three of the cube faces (sharing one mutual vertex) split by respective face diagonals,

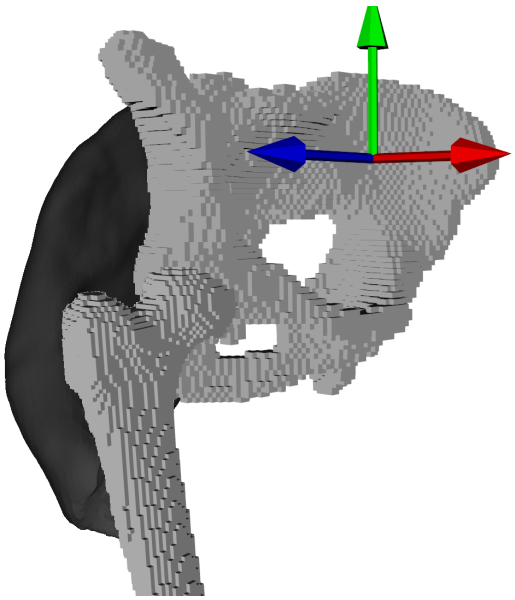

Figure 3.3: Volume mesh representation of bones, source: [KČ21]

producing **six** tetrahedra in total.

These structures can be obtained starting with the surface mesh, which can be, within its borders, filled with the primitives using e.g. the **flood-fill** algorithm.

On the one hand, these representations are instrumental in inferring various mechanical properties during their deformations, as even the density of the volume primitives (which can be heterogeneous) of the object can be expressed. On the other hand, they may require a lot more memory to be stored, and the visualisation methods are more complex.

## 3.2 Deformation methods

Regardless of the body representation, its behaviour during simulation can be viewed on the spectrum from rigidity to elasticity. **Rigid** bodies undergo only affine transformations and the relative distances between the vertices do not change. **Elastic** bodies, on top of that, can be deformed. Deformation occurs when the relative distances between the vertices change due to acting forces. In the simulated scene, bodies can exhibit varying degrees of elasticity and rigidity, reflecting different positions along this spectrum.

The acting forces can be classified into external forces that are incoming from the external environment (e.g. gravity or attachment to a rigid body) and the ones that are generated or propagated on the inside (or across the surface). Propagation of a force, in this context, means that the force spreads consequently through the volume or surface, while usually losing magnitude (depending on the underlying physics of the force and the attributes of the object) due to the dissipation of mechanical energy to thermal energy. In a way, this energy is to be lost in the surroundings.

The deformation usually starts with an external force, which spreads throughout the elastic body as an internal force. This force is received by the primitives (e.g. the vertices, the edges, or maybe the finite volume) it is applied to. Each primitive can have a propagation-related state associated with it (e.g. the heat at the primitive, or maybe its elasticity). After being affected by the external force itself, it then, based on its internal state, propagates the force to connected primitives (often these are the neighbours), which in turn pass the force wave-likely to their neighbours. This makes the elastic bodies arguably more difficult to model than the rigid ones, as each primitive has to be accounted for, as opposed to solving the force propagation globally for all e.g. vertices at once in a rigid body.

During the deformation, various **constraints** can be defined over the primitives, which can permit or cause the propagation of forces and dislocations under specific rules. In the case the constraint is permitting the propagation, one could imagine that e.g. neighbouring vertices can not be displaced too far away from each other, the volume of the finite sub-volume can not change, etc. In the opposite case, an example

of a constraint that generates an internal force could be that e.g. all primitives must move outwards the surface mesh due to the growth of internal pressure.

In the case where many elastic objects exist in the scene near each other, these objects tend to interact greatly, where the deformation of one object may trigger deformation in the other object and the objects may collide with each other a lot.

The spread of forces and constraints definition possibilities in an elastic (in other words, deformable) body (object) usually depends on its structure.

## 3.2.1 Influence of object representation

Whether the object undergoing deformation is mesh-less, a surface mesh, or a volume mesh allows different levels of information to be taken advantage of, often resulting in varying deformation results. For example, a representation implicitly containing topology information, such as the surface mesh, allows for constraint definitions directly on the pairs of positions defined by the edges, using the directions of the edges as the directions of internal force propagation.

### 3.2.1.1 Mesh-less methods

With no information about the structure, relying solely on the positions of vertices, the mesh-less objects must consider all pairs of vertices for force propagation (although using a pseudo-structure such as K-nearest pairs can also be employed), hence yielding the least computational efficiency of $\Theta(n^2)$ given $n$ vertices.

### 3.2.1.2 Surface mesh methods

The surface meshes provide information about the neighbourhood of the primitives, and can therefore consider only this neighbourhood during the propagation, not only cutting down on the computation time but also introducing well-defined spatial coherence (locality), which may hold more realistic results. On top of that, the direction of the force can now be estimated or computed using the information about the directions of the edges the forces are propagated across.

Deformation methods convenient for the surface meshes typically include:

- the **Laplacian Editing** (LE), which deforms the mesh based on constrained vertices and in between them, tries to preserve the local mean curvature and normals,
- the **Mesh Skinning** (MS), which parametrises the vertex positions based on the inner mesh skeleton,
- the **Cage-Based Deformations** (CBD), which parameterise the vertex positions based on the outer mesh skeleton,

- the **Mass Spring System** (MSS), where a set of elastic strings connecting the vertices is defined, constraining them in distance,

- and the **PBD**, where many types of constraints can be defined, e.g. preserving the volume, the vertex distances, the dihedral angles of the primitives, etc.

#### 3.2.1.3 Volume mesh methods

The volume meshes can provide the greatest detail, while also only propagating across neighbours. Adding depth information allows for even better force direction estimation and hence perhaps a more realistic force propagation. An example could be pouring boiling water into a metal cup. The cup becomes hot itself even on the outside very quickly, because not only can the heat be propagated on the surface of the cup through the collisions of the water with it (conduction), but the heat can also propagate through the volume of the cup, reaching the other side of the surface quickly (convection).

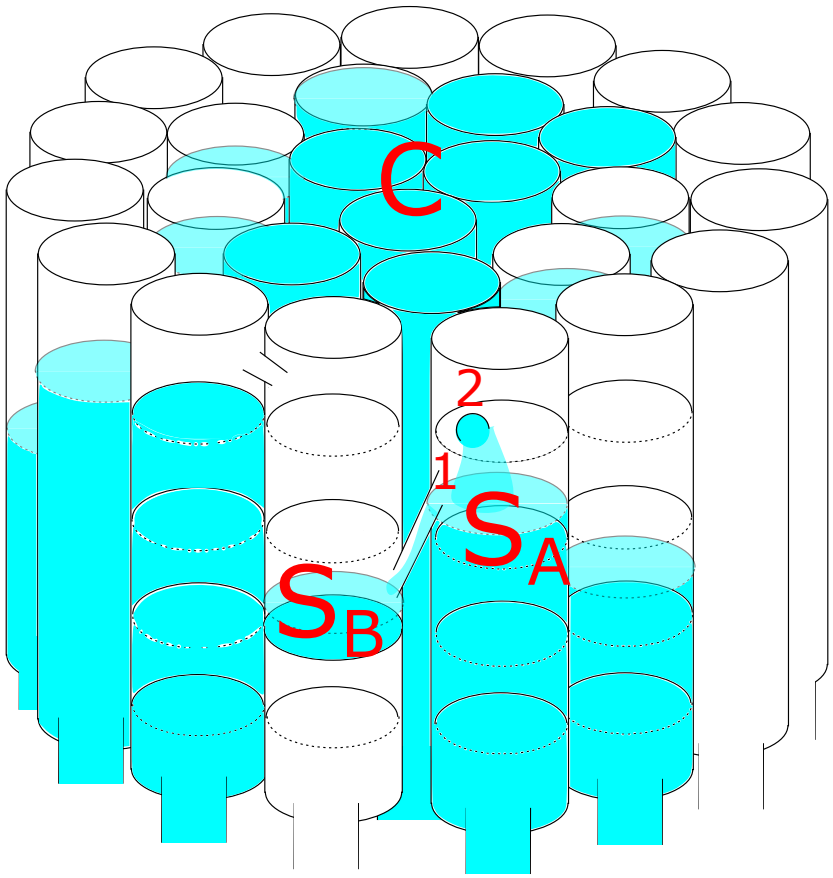


Figure 3.4: Tree water propagation model illustration

Compared with the surface mesh deformation methods, the constraints can also be defined on the inside of the mesh, typically concerning adjacent e.g. tetrahedra and hence more vertices.

Consider the example in Figure 3.4, which illustrates how water could be propagated through the trunk of a tree. The primitive, in this case, is each cylinder segment e.g. $S_A$ or $S_B$. The segments are illustrated only for the front four rods. From the bottom of the rods, water may permeate from the ground, depending on the distance from the $C$ centre rods (the central rods are filled the most). These are the inputs to the propagation model, analogous to external forces during deformation. Each segment holds a value of its fullness. If it is full and the water rises, it is propagated to the next upper segment, perhaps even in a non-linear relationship. The water is constrained not to fall through the segments. Moreover, the volume mesh allows for surface as well as volume propagation, which happens at the node $S_A$, where **(1)** the water is propagated to node $S_B$ on the surface, and **(2)** the water is propagated from one of the inner nodes to the node $S_A$, which could not be modelled this precisely using a surface model.

Besides, volume deformations can behave heterogeneously, where different primitive densities in different parts of the mesh could influence the magnitude of spread, practically allowing the 3D modelling of force sources, sinks, saddles, and spirals. For example, in a simulation of applying gravity to the *pelvis* with a prosthetic implant to support the bone, this implant's density and other mechanical properties could be modelled in contrast to those of the bone to hopefully find the optimal placement for it to prevent a fracture [Lob+22].

Some of the deformation methods applicable to volume meshes are:

- the **Finite Element Method** (FEM), where the object is broken down into finitely small elements, where each vertex holds local system information,
- and also the **MSS**, where the springs can also be defined on the inside,
- and **PBD**, behaving similarly to FEM while being computationally efficient.

The MS and CBD methods can also be used to deform volume meshes, since they deform the space itself, and are therefore representation independent, but probably mostly used in surface deformations.

### 3.2.2 Deformation method solvers

Consider the FEM applied to a rod with two vertices in just one dimension. For each vertex, an equation describing the sum of forces applied to it can be defined (one external force for the specific vertex and one internal reactionary force of the

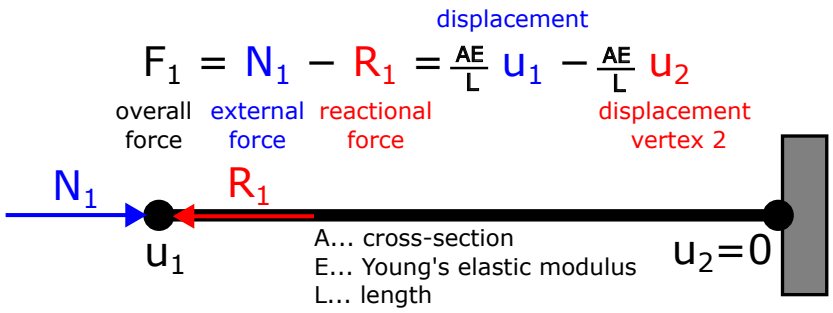


Figure 3.5: 1D FEM system with a unique solution

other vertex). Put together with some constants describing the mechanical properties of the edge, a system of linearised differential equations describing for each force, the distribution of displacements of each vertex. Since this system is singular (determinant of zero), infinite non-unique solutions exist (as long as the forces do not cancel each other out).

That is why boundary constraints must be introduced to the system, e.g. anchor one vertex in one coordinate system dimension. Then, this system has exactly one unique solution given either known force or displacement of the other vertex. Illustration can be seen in Figure 3.5, where a rod with the rest-length of $L$, the cross-section of $A$, and Young's elastic modulus $E$ is being one-dimensionally deformed by the external force $N_1$.

In the general 3D case, the task of the method solver is to find one solution to the system, but rather its approximation using an iterative numerical method. This solution is generally not unique and prone to errors of the numerical method or its lack of convergence. For example, imagine that given heat propagation on a one-dimensional rod, many distributions of heat across the in-between vertices would satisfy the two boundary conditions at the tails of the rod (more unknowns than equations, meaning the system is **underdetermined**).

Since the displacement problem is divided into small parts of the whole object problem, where the motion differential equation for each element is linearised, the solution to the displacement of one e.g. vertex is directly affected only by its neighbours and if added up, these local solutions have the potential to approach the global solution.

### 3.2.3 Specific methods

A brief overview of how the mentioned methods work follows.

#### 3.2.3.1 Laplacian Editing

Inspired by differential geometry, the Laplacian operator takes a scalar function on the input, computes its gradient, and then takes the divergence of the gradient, outputting once again a scalar function.

In a discrete setting, for **manifold** 3D surface meshes, the operator can be reformulated for a vector function (positions of the vertices flattened into a one-dimensional vector). It can be shown, that the divergence of the gradient must not be computed, as in the discrete space an equivalent can be expressed using the mean curvature and surface normals [Soro5]. For a manifold mesh, the operator is usually represented in the form of a spare matrix, which usually for each vertex contains a row of weights representing the contributions of neighbour vertices to the direction towards smoothing of this row vertex (sum of finite differences of the vertex with its neighbours). The weights may be just the inverse value of the vertex degree (combinatorial) or reflect the areas of the neighbouring polygons to overcome vertex distance irregularities, as is done in the case of the *cotangent weights* (or rather the mean-value coordinates, mimicking the cotangent weights while avoiding its drawbacks of negative values and large mesh problems) [Soro5]. Either way, this matrix expresses the localised geometrical relations between the vertices.

Given this matrix for the rest-position mesh, the mean curvature normals, expressing the details about local curvature for each vertex, and the constrained vertices (which are transformed by the environment), a sparse, a symmetric linear system emerges, which can be pre-factorized at the start of the simulation using e.g. the Cholesky decomposition and for each solver step, the solution for the vertex positions can be obtained in linear time via back-substitution [Soro5].

The main goal of this deformation is to keep the object surface smooth [Soro5], through the parameterisation of the vertices based on the linear system stemming from the discrete Laplacian operator. However, the details desired to preserve are not **rotation invariant** [Soro5], which may lead to distortions in the details during large rotation transformations. Figure 3.6 shows sample LE deformation result in the last row **(c)**, illustrating poorly preserved local curvature. The row **(a)** contains the original mesh, and the middle row contains the same constraints applied but using a rotation invariant deformation. The resulting deformation may also self-intersect, which has to be taken care of.

Interactions of the bodies deformed by LE do not initially seem feasible, as this approach mainly focuses on the preservation of local mesh geometry, seemingly ignoring the surrounding geometries in its essence.

However, this problem can be reformulated as a nonlinear energy minimalisation problem, as Huang et al. [Hua+06] describes. This problem can be solved using an interactive solver, where e.g. in between the approximations of the solution, the

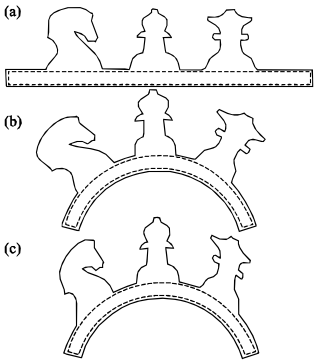


Figure 3.6: Detail preservation comparison of the original mesh (a) deformation by (b) a rotation-invariant method with (c) Laplacian Editing, source: [Sor05] (vectorised)

surface meshes can be tested and resolved for intersections. In such a reformulation, not only can more constraints be added, but even non-linear boundary conditions can be incorporated, which can be, for example, the preservation of the volume [Hua+06]. In addition, in the paper, the authors first solve the minimalisation problem on a coarse cage control mesh surrounding the original one and then interpolate this deformation onto it. This also allowed the authors to define the Laplacian in a rotation-invariant manner. Given these extensions, this method seems to be a reasonable choice for an interactive scene of deformable bodies.

#### 3.2.3.2 Mesh Skinning

Quite analogous to the musculoskeletal system, Mesh Skinning is a way of parametrising the positions of the *skin* (surface mesh) by a transformation of underlying inner *skeleton bones*. The bones can either be manually defined or obtained through mesh skeletonisation. The **affine** transformations (translation and rotation) of the bones are usually defined in a hierarchical system, as can often be seen in computer graphics scenes with objects (each bone has its local transform, but also the transform of the parent applied to it). A simple example of deforming an object with inner skeleton bone transformations using the Blender software (www.blender.org) can be seen in Figure 3.7.

Each of the surface *skin* vertices is influenced by one or more bones, in which case these bones usually have weights associated with them for the particular vertex. To obtain the final position of the parametrised vertex with more than one bone, **linear blending** can be used.

For vertex $v$, its new position $v'$ depending on the bone transformation, is defined in the subsequent linear blend skinning Equation 3.1 [Dom05].

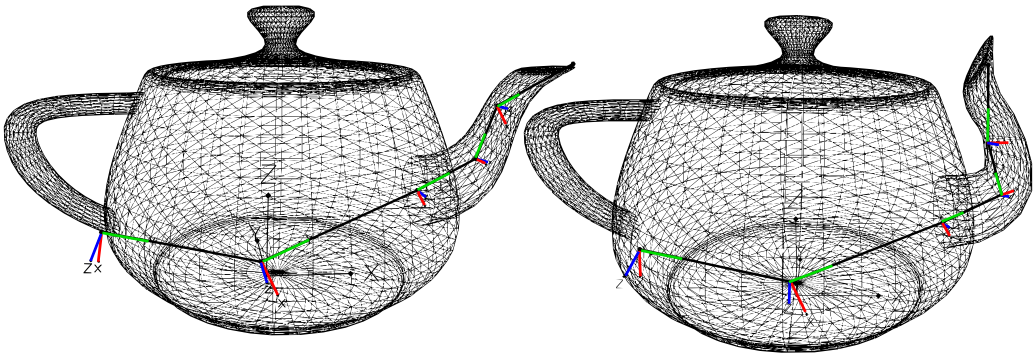

Figure 3.7: Mesh Skinning result using Blender

$$v' = \sum_{i}^{n} w_i M_i v; \quad \sum_{i} w_i = 1 \tag{3.1}$$

Where $n$ is the number of influencing bones, $w_i$ is the weight of bone $i$ and $M_i$ is the affine transformation matrix of bone $i$.

Since the skeleton can be transformed freely, it happens easily that the mesh starts to self-intersect, which may not be a big problem during e.g. animation done by an artist but has to be dynamically taken care of in the case of an interactive simulation.

The use of this method during interactive simulation also seems to be highly dependent on the quality of the underlying skeleton. Also, the weights of the bone influences must be properly tuned to achieve the desired results. Lastly, even the skeleton deformation, in case it is complex enough to underpin the behaviour, has to be somehow computed, if not done by hand. On the other hand, with the incorporation of e.g. bounding volumes to test for intersections, if the paths of well-built skeletons were known, relationships between them would be easily manageable, as the skeletons can provide a great level of simplification of this problem.

### 3.2.3.3 Cage-Based Deformation

This deformation also depends on a skeleton, only this time, **the skeleton is external**, or an **cage**. The only presumption needed to fulfil for this method to work is that the cage must contain all vertices and polygons of the deformed mesh [NS13].

A widely used method to implement the relation between the cage and the vertices on the inside of it is using the **Mean Value Coordinates**, namely the mean value theorem coupled with the harmonic coordinates theory [NS13]. The main problem is to approximate the solution to a harmonic function over the mesh while satisfying Dirichlet's boundary conditions [NS13]. Once the solution of the harmonic function is approximated by a piece-wise linear function, using the **Generalized**

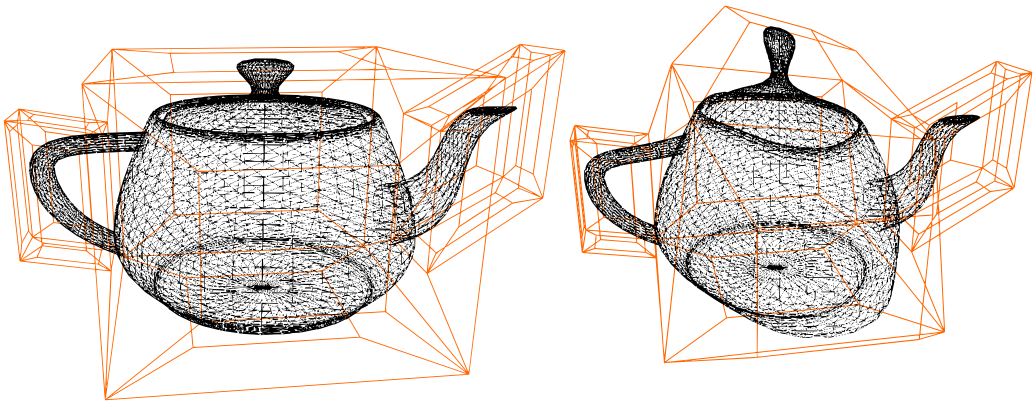

Figure 3.8: Cage-Based deformation result using Blender

**Barycentric Coordinates** concept, the deformations of the mesh can be obtained through a linear combination of the control points of the cage.

Once again, e.g. if the cage itself self-intersects, the inner mesh self-intersections must be accounted for. Moreover, obtaining how exactly should the skeleton be deformed would be an easy task for an animator, but a challenge to implement in an interactive, dynamic scene. But once again, the merit of this method for the interaction of many deformable bodies is that this problem could be simplified to the interaction of the outer skeletons, given the skeletons manage the underlying complex geometries well. The results for CBD can be seen in Figure 3.8.

#### 3.2.3.4 Mass Spring System

This first physics-based deformation method, an MSS, defines spherical particles (typically in the vertices of the mesh), holding information about the particle mass, radius, position, velocity, and even currently applied force. Pairs of particles can be connected by a spring of a particular stiffness. These springs often create a network, which is used to propagate forces. Sample MSS structure is depicted in Figure 3.9.

The main drawback of this method is the tuning of the stiffness and designing the network architecture since the method behaviour heavily depends on both.

Each spring represents a truly mechanical spring, hence solving of second order differential equation per adjacent spring is needed for the computation of displacements of one pair of vertices.

The method lacks constraints for shape and volume preservation. Moreover, defining the relationship of interacting bodies through this spring system would probably be even more architecturally dependent and hard to automatise.

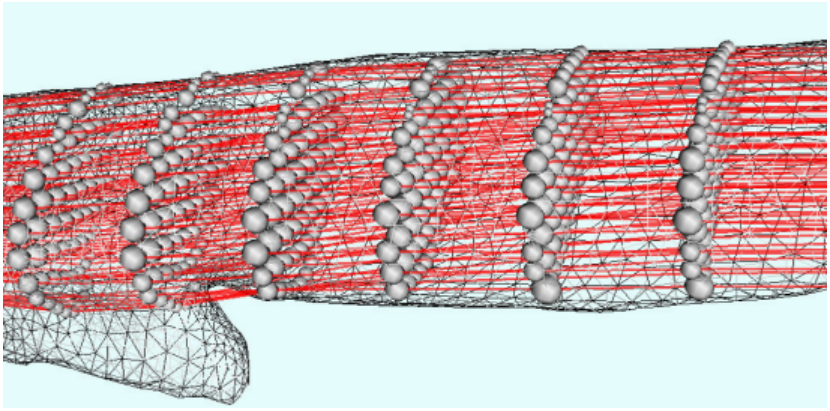

Figure 3.9: Sample MSS structure, source: [Jan12]

### 3.2.3.5 Finite Element Method

Another physics-based deformation model, the FEM, has been already briefly described when describing the solver in section 3.2.2. The method works by discretizing the (usually volume) mesh into finitely small elements, where the solution to the differential motion equations can be linearized and then the global solution can be built up from these fine elements.

The finer the discretization, usually the better the deformation results. These

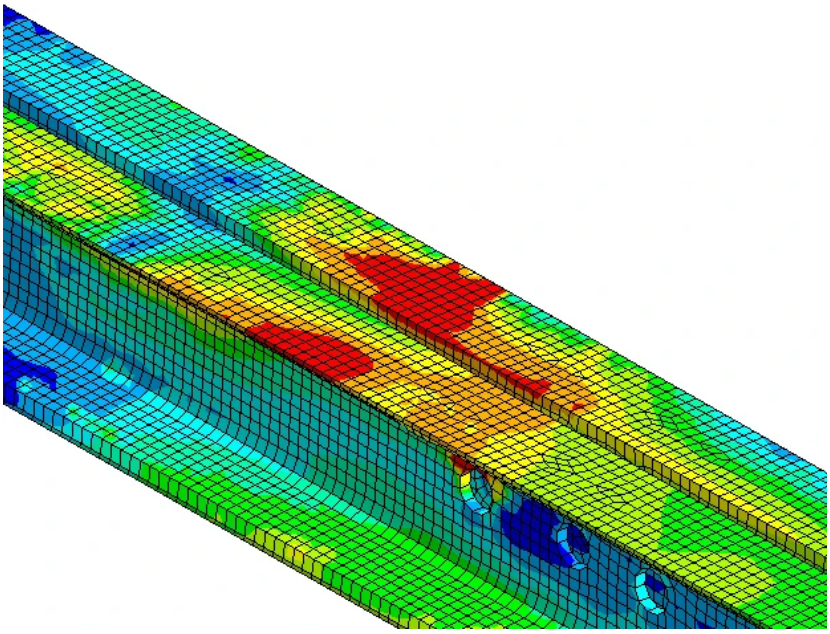

Figure 3.10: FEM structural analysis visualisation, source: featips.com/2022/09/23/the-basic-concepts-of-fea/

elements can contain various physical properties, such as heat, or other forms of energy, and propagate them to their neighbourhoods. Heterogenous areas can be outlined using the varying densities of the elements per volume unit to better reflect reality. The elements also have stiffness associated with them, which represents, crudely put, the resistance to the propagation of forces.

A huge pitfall compared to the surface deformation methods, even using modern acceleration methods, is the computational time expense. This is because the number of elements (and there of the equations) is usually much bigger. For example, given that a sphere has a surface of $4\pi r^2$ and a volume of $\frac{4}{3}\pi r^3$, then if sampled with primitives, the number of primitives for the volume would be roughly $\frac{r}{3}$ times more than the number of surface primitives of the same resolution. If computational complexity was not a concern, this method would probably be a great candidate for modelling the interactions of the elastic objects, as it allows a great level of flexibility in defining its constraints.

A visualisation of a structural analysis (propagation of forces) using the FEM is shown in Figure 3.10. If deformation were modelled in the visualisation, the most red parts would deform the most, while the most blue ones would probably deform the least.

#### 3.2.3.6 Position-Based Dynamics

In the year of 2007, the physics-based PBD deformation method was introduced by Matthias Müller et al. [Mül+07]. The method overcomes the drawbacks of the MSS by providing a means of volume preservation and great tuning controls [BMM17] similar to the FEM but much faster thanks to the use of fast explicit time integration solver schemes [BMM17]. Compared with the FEM, the element (particle) velocities are just approximated by a difference in consequent positions, omitting their physics modelling and solving directly for the positions. The process of applying PBD can be viewed as a quasi-static one [BMM17], where the forces are applied to the system, but in very small increments, allowing for the negligence of inertial forces. This makes the method a great candidate for modelling the interaction of deformable bodies, providing a comparable level of control over the physical properties as the FEM while being much faster in execution.

A PBD particle can be represented by an arbitrary primitive, usually though, the particle is either a sphere or just a point (vertex). Either way, the particle often has a **mass** (equal to one in the context of this thesis for simplicity reasons), a **position** in the coordinate system and a **velocity** associated with it.

The particles are subject to Newton's second law of motion, which relates the particle's acting forces to its mass and velocity. The solutions to this law can be approximated using a *symplectic* Euler integration step (or the generally more precise,

second-order in time Störmer-Verlet step), which considers only the current velocity instead of the starting one [BMM17]. This integration step is unconditionally stable due to the constraints, keeping the particles in physically valid positions, but it is also robust and fast. If the particles are dislocated greatly, though, the force propagation waves might be not constrained in one simulation step, which can lead to oscillations [Mül+07].

The free motion of the particles, if not fixed to an e.g. bone transformation, is restricted using *holonomic* (velocity independent) **PBD constraints**, based only on the positions of the particles. A constraint is represented either by a generally non-linear equation or by a generally non-linear inequation [BMM17]. The constraint is satisfied in case the equation is fulfilled or kept in inequality [Mül+07]. Each constraint has a **stiffness** parameter associated with it, describing its strength of influence. Together, the constraint (in)equations form a non-linear system of inequations, which is solved iteratively (*inner* iterations) using the Gauss-Seidel scheme [Mül+07] in each *outer* simulation step (rather than the locally linearized system of equations using the Newton-Raphson iteration [BMM17]). The constraints can be of various types, e.g. preserving particle distances, shapes of the polygons, the volume of a mesh, and the handling of collisions, and the order of solving these constraints should be constant since the stability of the system is dependent on it in the context of one outer iteration [Mül+07]. To overcome the instability, various **damping schemes** can be employed, e.g. the point velocity damping scheme or the Rayleigh dissipation potential [MMC16] for constraint damping. On top of that, the projection of these constraints must conserve the linear and angular momenta of the particles in the cases of constraints with finite stiffness (for example not the collision constraints) [Mül+07].

For a single constraint, the linear and angular momenta are preserved implicitly if the dislocations of the particles are along the constraint function gradient if all masses are equal [Mül+07]. This also solves the under-determination of the whole system [BMM17]. Due to the fact, that the solution to the (in)equation can be approximated with this gradient, it then has to be normalized and scaled by the function value (acting as an objective value or constraint violation) in the opposite direction, which forms *de facto* a gradient descend 3.2 to obtain the displacements of the flattened positions vector $\Delta\mathbf{p}$.

$$\Delta\mathbf{p} = -C(\mathbf{p})\frac{\nabla_{\mathbf{p}}C(\mathbf{p})}{|\nabla_{\mathbf{p}}C(\mathbf{p})|^2} \tag{3.2}$$

Where the $C(\mathbf{p})$ is the constraint objective function and the $\nabla_p C(\mathbf{p})$ is the gradient of this function with respect to the point position $\mathbf{p}$. For individual particles, the gradient is, instead computed only for the encompassing constraint particles by the Newton-Raphson iteration step 3.3, providing a prediction $\mathbf{p}_i$ for this generally non-

linear constraint instead of using Gauss-Seidel iterations, which can not handle the non-linearity [Mül+07]. Equation 3.3 also includes the stiffness term $\mathbf{k}_j \in [0; 1]$ for the specific constraint $j$.

$$\Delta \mathbf{p}_i = -C(\mathbf{p}_1, ..., \mathbf{p}_n) \frac{\nabla_{\mathbf{p}} C(\mathbf{p}_1, ..., \mathbf{p}_n)}{\sum_j^n |\nabla_{\mathbf{p}_j} C(\mathbf{p}_1, ..., \mathbf{p}_n)|^2} \mathbf{k}_j \tag{3.3}$$

The pseudocode 3.1 shows the algorithm of how the deformation iterations proceed. Initially, the positions in the last outer step are set to the rest-pose mesh, the velocities are initialized (usually to zero if the objects are not in movement in the beginning), and the inverse masses are set up (which in this case are always equal to one). As another preprocess step, the distance and constraint constraints can be generated for each muscle edge **(6-7)** and the volume constraints for each muscle mesh **(9)**. On the lines **(13)** and **(15)**, the semi-implicit Störmer-Verlet method is performed for the prediction of positions $\mathbf{p_i}$, while applying point velocity damping scheme **(14)**. The line **(13)** adds current velocities computed per Newton's second law of motion using the time difference between consecutive outer simulation steps $\Delta t$, the inverse masses $w_i$ equal to one in our case and the external forces $\mathbf{f}_{ext}(\mathbf{x_i})$. The line **(15)** uses this velocity once again weighted by the time difference (hence the second-order method) for prediction. Then, the collisions are generated (detected) for each particle, which in this case is the vertex. The collision detection **(16)** has the current position $\mathbf{x_i}$ with the predicted one $\mathbf{p_i}$ on the input. Then, the Newton-Raphson inner iterations commence, while projecting all constraints including collision constraints (collision resolution) to correct the predictions $\mathbf{p}_i$ **(20-21)**. On line **(26)** the velocities are approximated by the difference in positions over the simulation time step, and on line **(27)**, the current positions are substituted for the predicted ones. Lastly, **(29)** the velocities should be carried over the next iteration.

Source code 3.1: Pseudocode of Position-Based Dynamics outer iteration overview [BMM17]

```
1  for each vertex i do
2      initialize x_i = p_i, v_i = 0, m_i = 1
3  end for
4  for each muscle M do
5      for each edge e in M do
6          generateDistanceConstraint(e)
7          generateDihedralAngleConstraint(e)
8      end for
9      generateVolumeConstraint(M)
10 end for
11 loop
12     for each vertex i do
13         v_i ← v_i + Δt w_i f_ext(x_i)
14         v_i ← v_i · c_damp
```

```
15          p_i ← x_i + Δt v_i
16          generateCollisionConstraints(x_i → p_i)
17      end for
18      while i < solverIterations do
19          for each constraint j do
20              compute Δp_i using Equation 3.3
21              p_{i+1} ← p_i + Δp_i
22          end for
23          i ← i + 1
24      end while
25      for each vertex i do
26          v_i ← (p_i − x_i)/Δt
27          x_i ← p_i
28      end for
29      velocityUpdate(v_1,...,v_N)
30  end loop
```

### 3.2.3.7 Extended Position-Based Dynamics

The extension [MMC16] of the PBD method (XPBD) was created to overcome a well-known problem the PBD method had, which is that the constraint stiffness is non-linearly dependent on the simulation time elapsed and hence also the iteration count [MMC16]. This means, that as the simulation time grows to infinity, the constraints also become infinitely stiff in an unpredictable manner, changing the behaviour of the constraints. The same happens as the time step decreases [MMC16]. With the overcoming of this drawback, this method seems an ideal candidate for controlling the interaction among deformable bodies in a fast, robust, controllable, and **consistent** manner.

Time-dependent model behaviour is highly undesired in interactive applications, which should not be limited by time and most importantly, should not behave differently when e.g. the user does not interact for the next few seconds and then does the same interaction with the scene as if the simulation had just started, but the objects behave differently. Namely, with high constraint stiffness, the constraints become more aggressive, producing larger displacements, and the system may even start to oscillate. The first problem, simply put, stems from the scaling of constraints by the constant $\mathbf{k}_j$ (Equation 3.3).

In the paper, Miles Macklin et al. [MMC16] came up with a solution to this problem, which is to redefine the constraint projection 3.3 by incorporating a "well-defined concept of elastic potential energy" [MMC16]. On top of that, they introduced a Rayleigh dissipation potential to the system, which penalises constraints' influence on particles with high velocity.

Instead of the inner iterations **(18-24)** in 3.1, they propose incorporating a cumulative Lagrange multiplier $\lambda$ per constraint, which is always re-set when the outer iteration starts. The proposed inner loop is depicted in the pseudocode 3.2. The following equations omit the inverse masses of the particles, as they are all equal to one.

Source code 3.2: Pseudocode of eXtended PBD inner loop [MMC16]

```
while i < solverIterations do
    for each constraint j do
        compute Δλ_j using Equation 3.5
        compute Δp_i using Equation 3.6
        λ_{i+1} ← λ_i + Δλ
        p_{i+1} ← p_i + Δp_i
    end for
    i ← i + 1
end while
```

Equation 3.5 computes the multiplier change, where the term $\tilde{\alpha}_j$ represents time-scaled **compliance** defined by Equation 3.4. The compliance itself is the multiplicative inverse of this constraint's stiffness ($\alpha = \frac{1}{\mathbf{k}}$). The term $\lambda_{ij}$ represents the so far cummulated Lagrange multiplier for this particular constraint $j$ at iteration $i$ [MMC16].

$$\tilde{\alpha}_j = \frac{\alpha}{\Delta t^2} \tag{3.4}$$

$$\Delta\lambda_j = \frac{-C_j(\mathbf{p}_i) - \tilde{\alpha}_j\lambda_{ij}}{|\nabla C_j|^2 + \tilde{\alpha}_j} \tag{3.5}$$

$$\Delta\mathbf{p}_i = \nabla C_j(\mathbf{p}_i)^T \Delta\lambda_j \tag{3.6}$$

The Lagrange multiplier serves as a regularization term, and if the compliance is set to zero, this term also becomes zero and what is left is the original Newton-Raphson update defined in Equation 3.2 with the constant scaling [MMC16]. This makes sense since raising the stiffness to the infinity brings the corresponding compliance to zero. From this observation, a drawback emerges, which is that the stiffer the constraints are, the more this method resembles the original PBD.

Another merit of this method is an additional constraint-damping scheme. This scheme reflects the **Rayleigh dissipation potential**, which lessens the impact of constraints with fast particles. This means, in a sense, letting the particles *finish* a greater movement before constraining them.

With just the addition of one parameter $\beta$ and its time-scaled version $\tilde{\beta} = \Delta t^2\beta$, used for tuning of this damping coefficient, and also the incorporation of current particle velocities, the damping of constraints changes the Lagrange multiplier update as follows (3.7 [MMC16]), once again with the masses omitted. The $\nabla C_j(\mathbf{p}_i - \mathbf{x}^n)$

term represents the change in position compared with the last iteration, in other words, the discrete approximation of the velocity.

$$\Delta\lambda_j = \frac{-C_j(\mathbf{p}_i) - \tilde{\alpha}_j\lambda_{ij} - \gamma_j\nabla C_j(\mathbf{p}_i - \mathbf{x}^n)}{(1+\gamma_j)|\nabla C_j|^2 + \tilde{\alpha}_j}; \quad \gamma_j = \frac{\tilde{\alpha}_j\tilde{\beta}_j}{\Delta t} \tag{3.7}$$

## 3.3 Deformable collision handling

A collision occurs when two bodies (objects) collide. The collision starts when either a vertex or an edge starts entering the volume of the other object, up until a whole triangle intersects it. Depending on the desired precision, a test for collision on all of these primitives can be made. During a real-time interactive simulation, it usually suffices that the vertices do not penetrate other objects, as long as the polygons do not get excessively degenerated (elongated), so much so that the edges start to penetrate, while the vertices do not. Such a situation can occur when so-called tunnelling occurs. Tunnelling can occur for various reasons. The main reason is that during a discrete simulation, the geometries may change so rapidly, that between two consecutive steps, a primitive completely tunnels through the other object, with no implicitly obtainable piece of information about it being inside of the object. Objects intersecting is an unrealistic state of the model and should not be present.

Collision handling consists of two phases. Firstly, the collision has to be detected either *a priori* (before it even occurs) or *a posteriori* (after it occurs). The two paradigms are usually described as continuous (as in precise prediction of collision position on a continuous trajectory) and discrete (as in only concerning discrete positions of the primitives), respectively. Secondly, the detected collision must be either avoided (*a priori*) in a time window or resolved (*a posteriori*) by generating the collision forces.

The avoidance is straightforward in case the exact position of an intersection on the other object surface is known (either with a polygon, edge or directly a vertex), as the primitive can be set to that position or slightly before it on the trajectory e.g. to avoid the intersection of its bounding box, in case the primitive has any dimension, as opposed to a dimension-less vertex, which has no surface nor volume, and its position directly on the surface of the other object can be deemed valid. This approach may better reflect the reality but is usually harder to implement and more computationally expensive.

In case no such precise position of collision is computed, the collision must have already happened at the time of (discrete) detection. The collision forces to escape the intersection may be generated from the information about the direction of the collision, the velocities of the primitives, or based on the surface or internal friction of the colliding objects. Additionally, the position of the primitive before and after the

collision may be known, sometimes containing intersecting local bounding boxes, the direction and distance of the colliding point to the surface may also be known, etc. However, utilising the detailed information can be a lengthy process. Therefore, during real-time interactive simulations, some object properties can be omitted, e.g. the physics-intensive calculation of friction, or perhaps the self-intersections. Depending on the desired precision, the objects may intersect by a small amount, while hoping to be resolved at least in the future.

The collisions must not be necessarily handled on the computational model, but a more convenient, approximating collision model can be employed. Such a collision model can represent dimensionless primitives with e.g. cubes with the same orientations or spheres, which are both suitable for fast collision checking since all that is needed to test is their distance. 2D polygons, such as triangles, are often not feasible since too many colliding configurations among such primitives exist, differing in positions but also orientations. Once collisions are handled on the collision model representation, the resulting deformations should be propagated to the perhaps finer detail computational model and any other representations the scene may contain (e.g. the visualisation layer may differ from the computational one). For example, in the game industry, it is often the case that the collision model is just a crude mesh bounding the object (e.g. A convex envelope or maybe the bounding box).

In a scene with many objects, collision detection is further split into two phases. The first, broad phase, swiftly prunes the space for candidate objects that might collide based on e.g. the bounding box overlap. Then, for these candidates (maybe a small subset of all the objects), a narrow phase commences. The narrow phase considers the details of the geometries, e.g. at the level of the primitives.

Compared to the collision handling of rigid bodies, elastic bodies can often intersect at many places at once, and generate considerably more self-intersections. Hence, various acceleration approaches are often advantageous. However if, for some reason, the precise primitive-to-primitive collisions need to be handled, just the narrow detection phase becomes a great time complexity bottleneck.

### 3.3.1 Continuous collision detection

As previously declared, many mutual configurations of two arbitrary primitives may exist. One, computationally expensive option, is to perform the *a priori* collision detection by advancing a trajectory of the primitive from the last time step to pinpoint exactly where it intersects with the other primitive. A simplified, 2D example with the primitives of a dimensionless vertex and an AABB of a triangle is shown in the following Figure 3.11.

In the example, a black triangle of one object on the right is advancing position

to the blue triangle. One of the new positions in the vertex $P$ starts to collide with an AABB of the other, red mesh on the left. The collision is firstly detected in the broad phrase, as the point $P$ enters the bounding box, and continues in the *a priori* precise hit detection. In this most basic approach, the trajectory of the colliding vertex is discretely approximated by advancing the previous position towards the new position $P$, until the boundary is met. Borrowing from traditional rendering techniques, this method can be accelerated by using e.g. the **Ray Marching** method (in the case the signed distance field of the other objects is available). For general convex primitive configurations, techniques, such as the **Gilbert-Johnson-Keerthi** algorithm using the **Minkowski difference** [Gui09] are used to quickly determine the intersection's existence and even the closest pair of points on the surfaces of the primitives together with their distance.

## 3.3.2 Structure-based methods

Given complex configurations of geometries that may emerge from interactive simulation with deformable bodies, collision handling, mainly the detection phase, needs to be accelerated taking advantage of diverse structures. The generation and updating of these structures is of great importance [Tes+05], as it is usually the bottleneck of the method, while the geometries of deformable bodies might change very often. The structures vary in the collision information provided, but also in their flexibility to update.

The structures are usually built per object, which they virtually partition into sub-spaces, where alienated sub-spaces can be quickly rejected for collision detection since the contained geometries have no way of intersecting. This is analogous to the broad phase, only this time, not only do the objects define the sub-spaces but so do their sub-parts.

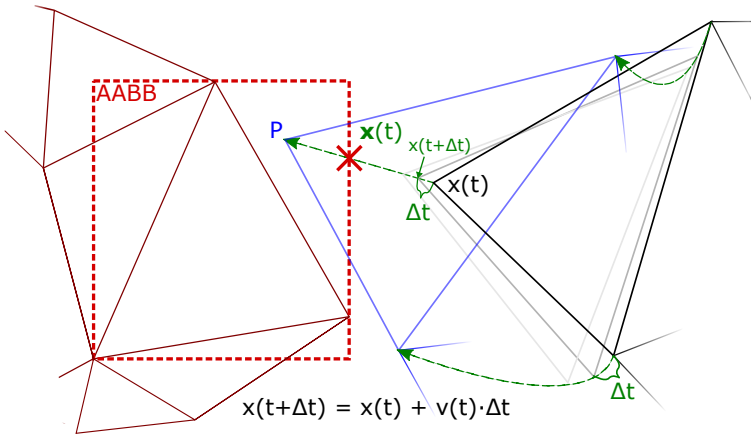


Figure 3.11: Point trajectory *a priori* intersection with a triangle AABB boundary

#### 3.3.2.1 Binary space partitioning

One of the simplest ways of partitioning the space is using binary space partitioning. This method uses hyperplanes to recursively subdivide the space into two sub-spaces until the smallest number of primitives is contained in the hyperspace (e.g. 2), then that node becomes a leaf. The partitioning is intuitively represented by a binary tree, hence the name Binary Space Partitioning tree (BSP tree). Depending on if the tree is balanced, the query for an arbitrary primitive for collision may be $\Theta(logn)$ on average or $\Theta(n)$ of the tree has its maximal possible depth (given $n$ dividing hyperplanes as the tree nodes). The same computation time complexities stand for the insertion or the deletion of a hyperplane, which are both needed as the primitives may change positions relative to the hyperplanes quite often. The construction of the hyperplanes may respect various criteria such as primitives spatial distribution, or perhaps symmetry features.

#### 3.3.2.2 Bounding volume hierarchies

These structures are also commonly kept in a tree structure, hence the name Bounding Volume Hierarchy tree (BVH tree). The tree is also constructed recursively, only by encompassing the primitive sets in their bounds. The bounds may be represented by spheres (figure 3.12), cylinders, or capsules, but most commonly by the axis-aligned bounding boxes (AABB), which are especially feasible due to their easy intersection tests of distances, but also due to efficient updating [Tes+05]. An arbitrarily oriented version of AABB is called the oriented bounded box, which can fit the primitives more tightly. Compared to partitioning the space with hyperplanes, the bounding boxes should surpass a BSP tree if the geometries are ill-posed, depending on the sophistication of the hyperplane construction.

An obvious remark is that if a tree is built for each object, while there are more objects in the scene, this structure becomes a forest, as with any other tree-like space partitioning per object. The trees may be appropriate to be built on top of different primitives and the trees may also be of a different type (octree, n-ary tree, kd-tree, etc.), which can result in a quite complex structure to represent [Tes+05].

The probability of collision increases for the child nodes if the parent node indicates so. Thanks to this, the search can be further accelerated by firstly searching bottom-up in the upper half of the tree, and in case of a potential collision, the particular sub-tree can be inspected in a top-down manner [Tes+05].

#### 3.3.2.3 Spatial hashing

A marginally different approach, although also partitioning the space into subspaces, is spatial hashing, which uses a hash function to map (for example uniform) grid

Figure 3.12: Bounding volume hierarchy using spheres on a triangular mesh, source: [Gui09]

voxels to a hash table [Tes+05]. This approach is especially feasible due to its easy implementation compared to the complex hierarchical structures, memory efficiency and flexibility in the form of possibly irregular grids. The grid resolution has been found to work the best when one voxel is approximately the size of the primitives [Tes+05]. This method is designed for volumetric meshes, with the typical primitive of a tetrahedron.

The spatial hashing table is constructed as follows. First, all object vertices are mapped to the has table. Then, it also maps every volume primitive to the grid cells it (more generally, its AABB) touches. To check for the collisions, the hash table entry corresponding to the query primitive is looked up, which may contain a polygon. If it does, a narrow-phase collision handling follows using the barycentric coordinates comparison, which also outputs the depth of the penetration. This table can also be used to detect self-intersections, as the table contains vertices as well as the polygons of the mesh, then, if both a vertex and the polygon in one entry are of the same mesh, self-intersection occurs [Tes+05]. A simplified 2D illustration of spatial hashing for circles can be seen in Figure 3.13.

The time efficiency mainly depends on the grid resolution and the hash function

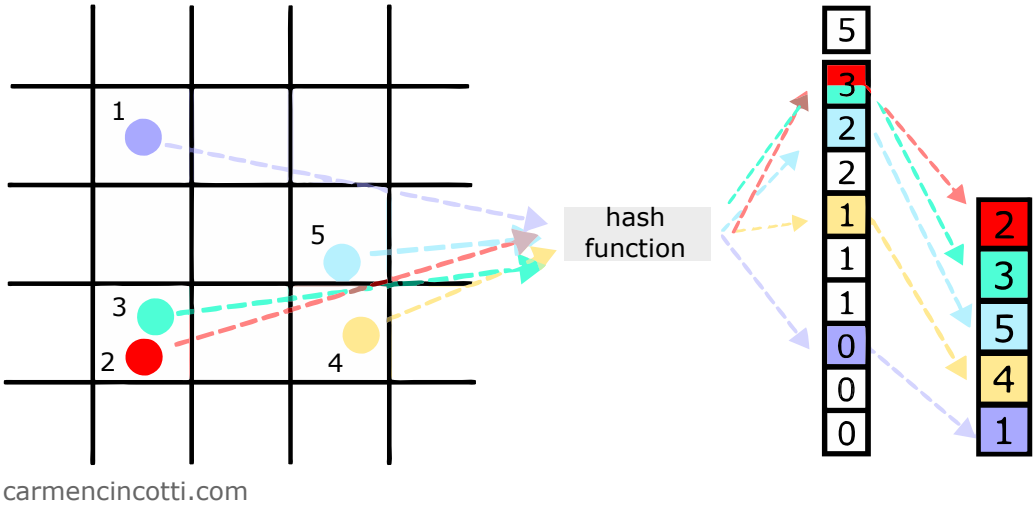


Figure 3.13: Simple 2D spatial hashing for spheres, source: carmencincotti.com/2022-10-31/spatial-hash-maps-part-one (vectorised)

distribution. Small hash tables may produce hash collisions, which would hinder the performance. The merit of this method is that it is independent of the number of the objects and their shapes [Tes+05].

#### 3.3.2.4 Signed distance fields

A **signed distance field** (SDF) is created by either discretizing the space on a single object into a uniform grid or may be kept in a tree structure, like the BSP tree. Considering the uniform grid, it samples every grid vertex and computes its distance to the surface of the object. The distance field is signed, if it also stores the information about the distance to the surface from the inside or outside the object in the form of a sign of the distance. For an arbitrary query point, its values are interpolated from the nearest sample vertices of the grid, which in case of a collision gives useful information to resolve it (the gradient to the surface). The discretization may also be adaptive to the level of detail of the underlying object [Tes+05].

Generally, these fields are not suitable for representing interactive deformable objects but are especially efficient when representing the collision model of a rigid object. This can be taken advantage of to handle collisions between deformable and rigid objects without the need for field updating (since the rigid object only undergoes affine transformations).

### 3.3.3 Stochastic methods

Due to the already discussed discrepancies between a computer model and a real object, it may not be fully reasonable to detect all collisions and resolve them, since

the surfaces are just mere approximations of the original shapes. Instead, collision handling that is not precise, may sometimes miss some collisions, but handles at least the majority of them, while being able to resolve the others perhaps later in the simulation, and is often probability-based, hence the name stochastic methods.

Moreover, the missed collisions (for visualisation) may not be even noticeable by the observing person or may not affect the computational model significantly (during knowledge inference). Often, the more important thing for the user is whether the deformation is fast [Tes+05].

Stochastic approaches may use the BVH trees to predict only the probability of collisions of two leaves but never check the actual primitive-to-primitive collisions, which is usually the bottleneck of a collision handling method. Other stochastic approaches may try to prevent collisions by pseudo-random sampling the subspaces while also operating some heuristic criterion, such as the distance of the two sample leaves.

A stochastic method does not even have to utilise a particular complex structure but can express the surface relationships just through sample edges between them while measuring their lengths. The less the length of an edge connecting two surfaces, the more the probability the connected primitives may collide in the future. Such a method could be described to avoid the collisions *a priori* while operating with discrete values, without the trajectory-primitive intersections tests.

# Previous work 4

In the context of the **Muscle Wrapping 2.0** project gitlab.com/besoft/muscle-wrapping-2.0) which focuses on generating muscular lines of action wrapping around bones to better estimate various biomechanical properties of a musculoskeletal system, a surface muscle mesh deformation method based on the PBD method had been developed in the C++ programming language. The method, apart from implementing the constraints to preserve vertex distance, the dihedral angles of adjacent triangles and preserve individual volumes of each muscle mesh, also accounts for the anisotropy of the muscles (vertex distances on the edges along the direction of the underlying muscle fibres are stiffer than the ones running perpendicular).

## 4.1 System

The modelled musculoskeletal system contains a set of rigid bones and deformable muscles. All these objects are represented by a triangular 3D surface mesh, which may be a more complicated representation for the muscles as opposed to the commonly used one-dimensional Hill-type structures, which are geometrically polylines running from the muscle's origin to its origin representing the so-called *lines of action* [KČ21]. But this complication of representation serves a concrete goal, to provide a more realistic basis for the estimation of the mechanical properties of the muscles, as the simple *lines of action* often penetrate the bones and produce errors (up to 75% using just straight lines) [KČ21].

On the input, apart from the bone and muscle meshes, the bones should also have a movement associated with them to be executed during the simulation and the muscles should be supplied with the estimates of their origin and insertion areas, each defined by a sequence of points enveloping that area. Moreover, internal muscle fibres can be generated, represented by a set of polylines, usually starting at the origin and ending in the insertion areas, using e.g. the Kohout & Kukačka [KK14] method [KČ21].

In the system, the muscle points contained in the origin or insertion areas are transformed with the bone which contains the attachment site, and unaffected by the

deformation (other than being fixed to the bone). The other group of muscle vertices are pronounced to be the PBD particles, which are manipulated by the solver. This approach is called the **inverse kinematics** [KČ21], where the displacements of the muscle vertices are inferred from the locations and movements of the bones at all times, whereas in reality, the muscles cause the movement of the bones [KČ21].

## 4.2 Solver

As per the original PBD article [Mül+07], the solver is implemented as a Gauss–Seidel-type iterative solver, following the algorithm presented in section **3.1** in the original paper. This approach defines a PBD particle as a dimensionless, triangular mesh vertex with associated mass, position, and velocity. The method also uses point velocity damping. Aside from the **anisotropy** distance constraint stiffness modulation, no fundamental changes to the algorithm or constraint calculation 3.1 are made.

The modulation of the stiffness $\mathbf{k}_i$ for the edge $i$ is described by Equation 4.1, where the $\mathbf{u}_i$ is the **normalized** direction of edge $i$ and $\mathbf{v}_i$ represents the tangential direction normal vector of the nearest surface fibre [KČ21]. Due to the dot product, the resulting stiffness becomes zero if the vectors are collinear, in case the vectors are perpendicular, the result stays at one [KČ21].

$$\mathbf{k}_i = 1 - \mathbf{u}_i \cdot \mathbf{v}_i \tag{4.1}$$

The deformations of muscles are initiated by some of the muscle vertices being fixed to the bone attachment areas (origin or insertion). The displacement of these vertices, following the bone transformation, initiates primarily the vertex distance constraints in the neighbourhood, starting the wave-like propagation of forces also due to the other constraints.

## 4.3 Collision handling

Collisions are handled only for the collisions of muscle vertices to the rigid bones. For each bone, an SDF is created using the **Discregrid** library for generating SDFs for bounded meshes (github.com/InteractiveComputerGraphics/Discregrid). The process of creating an SDF starts with discretizing a subspace in the scene defined by the bone's bounding box, where an optional margin can be added. The resolution of the discretization is user-defined. Each voxel is represented by a 32-node Serendipity type. Then, its distance and direction to the closest bounded surface are computed for each of these nodes. The surface, represented by a triangle mesh, provides triangle normals to determine the shortest distance in case the closest point is inside a triangle. Special cases of the nearest point being directly on the edge or directly in one of the vertices are accounted for. The process of creating the field is

quite computationally expensive, depending on the resolution, which is generally desired to be high. This is why the collision handling is done only for the bones to satisfy the desired quality of the deformation to run ideally in real-time.

Once the discrete grid is generated, an arbitrary point in the subspace can be queried. The containing voxel is found for this arbitrary point, and the distance with the gradient is interpolated from the nearest nodes using a cubic Lagrange polynomial. If the mesh is also at least watertight (edge and vertex manifold, without self-intersections), a sign is given to the distance which represents whether the query point is on the inside or the outside of the bounded mesh. This interpolation holds a constant time evaluation.

Attempts to solve the **tunnelling problem** utilising the SDF have been made ([Čer+23]). To remind the reader, tunnelling occurs when the motion of one object or its part is so fast that it passes through the whole volume of the other object in one single simulation step, rendering the collision implicitly undetectable. In this particular case, a muscle object vertex would tunnel through the rigid bone surface. To detect such an event using the SDF, a simple and fast comparison of gradients towards the surface of the previous and current muscle vertex positions can be made. If this gradient changes significantly (e.g. by more than 135°), the directions start to point towards each other, which means that the particle changed its orientation relative to the bone surface, hence vertex tunnelling must have occurred. If the first direction is denoted $\mathbf{d}_i$ in iteration $i$ and the previous direction vector is defined as $\mathbf{d}_{i-1}$, the tunnelling test is described by Equation 4.2 [Čer+23].

$$arccos \left| \frac{\mathbf{d}_i \cdot \mathbf{d}_{i-1}}{\|\mathbf{d}_i\| \cdot \|\mathbf{d}_{i-1}\|} \right| > 135^\circ \tag{4.2}$$

The SDF is only computed once (at the start of the simulation) and does not get translated or changed, as this is a very costly operation. Instead, during the simulation, each muscle vertex is first transformed (by the bone's inverse transformation matrix) to the stationary bone space, where the SDF is, queried for collision (which occurs if the distance is negative), and in the case of collision, the resulting gradient representing the direction to the surface is transformed back to the moving bone space, where it is added to the vertex's positions, which is hence effectively pushed hopefully outside the bone.

## 4.4 Critique

The main pitfall of this model is that it is implemented as a *pure* PBD method, while the advanced XPBD method exists. This causes the constraints to approach infinite stiffness in time, causing progressive changes in the deformation behaviour as the simulation continues. This is not a preferred behaviour of the model, as it can not be

examined thoroughly since tuning of the stiffnesses only really takes an effect at the beginning of the simulation. However, studying the model's behaviour in a greater time scale allows for the inspection of how exactly the constraints influence the deformation, allowing further improvements.

Another drawback is the usage of a very small iteration count to satisfy the computational time efficiency goals. Although these few iterations (3) may yield somewhat reasonable results for some muscles, as shown in the [KČ21] paper, the further examination done as semester work for the course KIV/OP showed that most of the constraints exerted very little impact on the deformation, with no room for proper expression. Rather, the deformation of the vertices was found to be mainly driven by just the Störmer–Verlet approximation of Newton's equations of motion. The solver with these few iterations seemingly has no time (not enough iterations) to properly react to the fixed vertices displacements, resembling more of a cloth or fluid behaviour in its rapidity of movement and oscillations.

The software architecture also falls off as the deformation is implemented without the convenience of e.g. expression templates the modern mathematical libraries provide. On top of that, the constraint projections are mostly implemented only in their computational forms provided in the appendix of the [Mül+07] paper. This makes further modifications (such as introducing regularization terms) very hard, although not impossible.

The deformation was tested on quite a small dataset containing 4 muscles of the *glutei maximus*, *medius*, the *iliacus*, and the *adductor brevis*. Out of these muscles, the results are acceptable except for the *iliacus*, which, during the hip flexion, gets **passively dragged behind the bone and unrealistically broken in shape near its insertion area**, as is shown in Figure 4.1.

Last but not least, the **collision between the muscles**, whose interaction becomes especially relevant when adding more muscles near each other, **is not addressed at all**.

One of the hypotheses of this thesis is that **the modelling of active muscular interactive contractions should fix the unnatural bending of the *iliacus***. Due to the comparison between the PBD and the XPBD, this thesis also aims to implement the deformation using the novel, **XPBD method**, and lastly to **quickly and correctly detect and resolve the collisions between the muscles**.

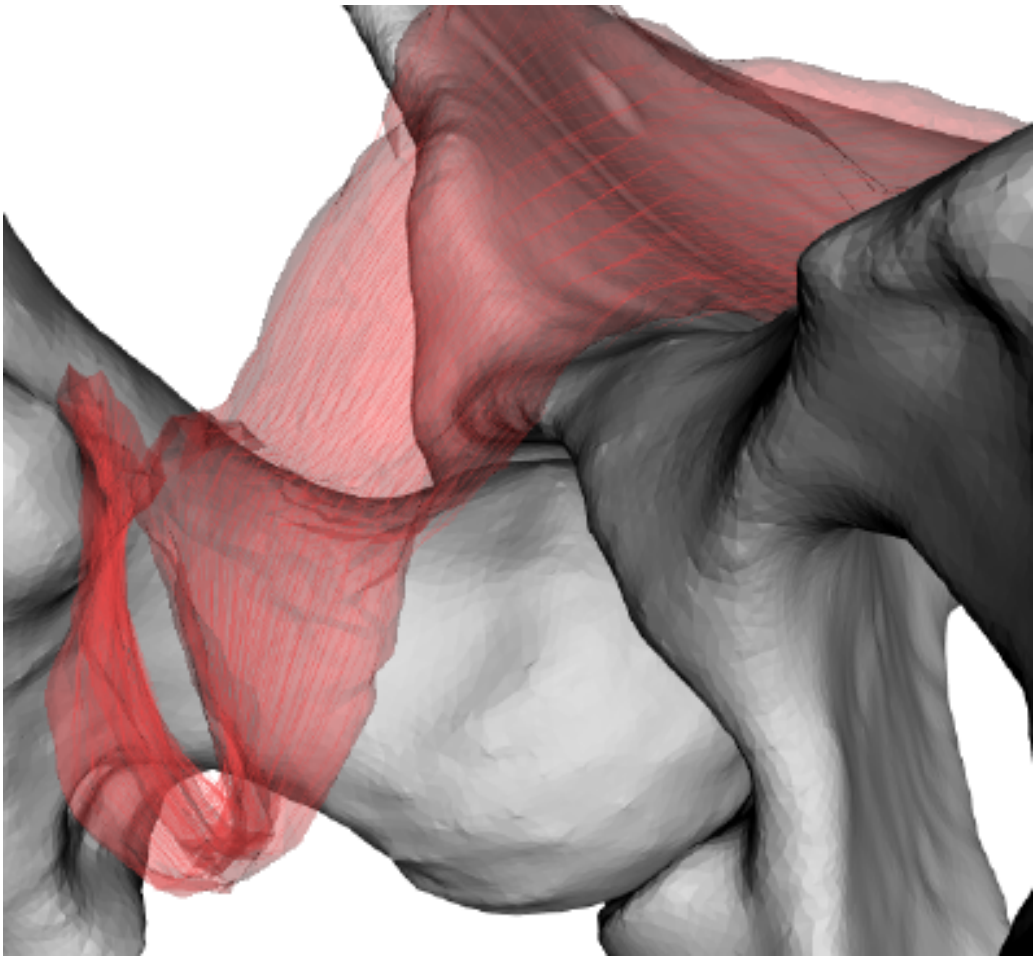

Figure 4.1: *Iliacus* unrealistically bending during hip flexion

# Proposed solution 5

This thesis proposes solutions to two problems of the current PBD muscle deformation model [KČ21]. The first problem is the **penetration of elastic muscle objects in the scene**, rarely addressed in the state-of-the-art musculoskeletal models, usually resulting in unrealistic states of the complex muscle geometries. The second problem is the **absence of active intermuscular interactions facilitated through governed synchronous activations (contractions) of the muscles**, currently rendering the state-of-the-art muscle models mere passive followers of the bone movements.

The first problem could presumably be interpreted as unrelated to the muscle deformation itself, as the forward simulation does not contain explicit knowledge of where the complex muscle geometries could lie further from the current step. In this light, the problem of muscle penetrations could be re-defined as a general problem of avoiding penetrations of dynamic elastic bodies, and therefore applicable to a broad spectrum of elastic body simulations. On the other hand, the muscles avoiding penetrations is indeed a form of a **passive intermuscular interaction**.

The solution to the second problem aspires to extend the concrete PBD model, exploiting the capabilities of this fast and robust approach and to design and implement an approach to facilitate **active muscle interaction** in contraction during various movements around the hip joint. To achieve this goal, though, fundamental changes to the current PBD model have to be made.

The first of the changes is to extend this PBD model to its extended version, called XPBD. There are many reasons for this change. The XPBD method had been developed to overcome a well-documented shortcoming of the PBD method of infinite stiffness and is, in consensus, deemed its successor. The infinite stiffness problem describes the stiffness of the constraints to theoretically reach infinity in the infinite time step of the simulation. That is the function of the constraints changes during the simulation and is therefore time-dependent (and thus iteration count dependent).

The time-dependency of PBD poses a difficult challenge in correctly assessing model behaviour in terms of the currently implemented constraints, as well as the

to-be-designed muscle interactions. This is simply due to the reality that the more iterations the solver is provided, the greater the chance for the solver to reach global optimum, and the more pronounced individual constraints have a chance to be, but also the more inconsistent the constraints become.

The need for more solver iterations also calls for a more efficient implementation enjoying the parallelism potential of the PBD method as well as the use of modern vectorizable mathematical libraries such as **Eigen** (eigen.tuxfamily.org) mainly because of the requirement of the theoretical model to run in real-time, efficiently. This requirement stems from the proposition, that the scene should be interactive, and therefore responsive, as the user may rate the quality of experience as worse in a scene which is precise but slower in contrast to a better experience of a simulation, that is real-time and responsive [Tes+05]. Moreover, no specialised hardware should be needed to run this software, as it is meant to enhance the decision-making done by medical experts, who usually have no access to such hardware.

As these changes to the solver are a form of preparation of the environment for the development of the active intermuscular interaction, they will be the first to be described, followed by the main proposals of this thesis.

## 5.1 Extended Position Based Dynamics

Compared with other deformation methods, the XPBD is still relevant in the area of deformable bodies modelling thanks to the robustness, speed of evaluation, and good control of the model parameters [BMM17]. Additionally, the **Rayleigh dissipation potential** constraint formulation should be implemented, as it provides additional damping force for the price of just a few, cheap, mathematical operations [MMC16].

As the authors stress [MMC16], the extension from PBD to XPBD should be straightforward. But the method described in the article [KČ21] has many architectural backdraws, preventing this easy transition. For example, the constraints are not represented by an instance of an object, which could hold the state of the constraint, such as the Lagrange multiplier, and their projection is implemented in the computational form from the original paper [Mül+07] for each constraint type independently, making modifications to the projection equation hard to pinpoint, as opposed to the alternative, where this projection would be implemented just once for all constraints (as it is mostly the same). On top of that, a mere extension would make comparisons difficult to navigate and cluster the code base. This is why, a separate XPBD deformation algorithm should be defined in the **Muscle Wrapping 2.0** project. The general control flow graph of both methods stays roughly the same, which in the case of the XPBD, will be as depicted in the scheme in Figure 5.1, with some details omitted for clarity, e.g. all constraints should also be created in the **init** method of the XPBD instance.

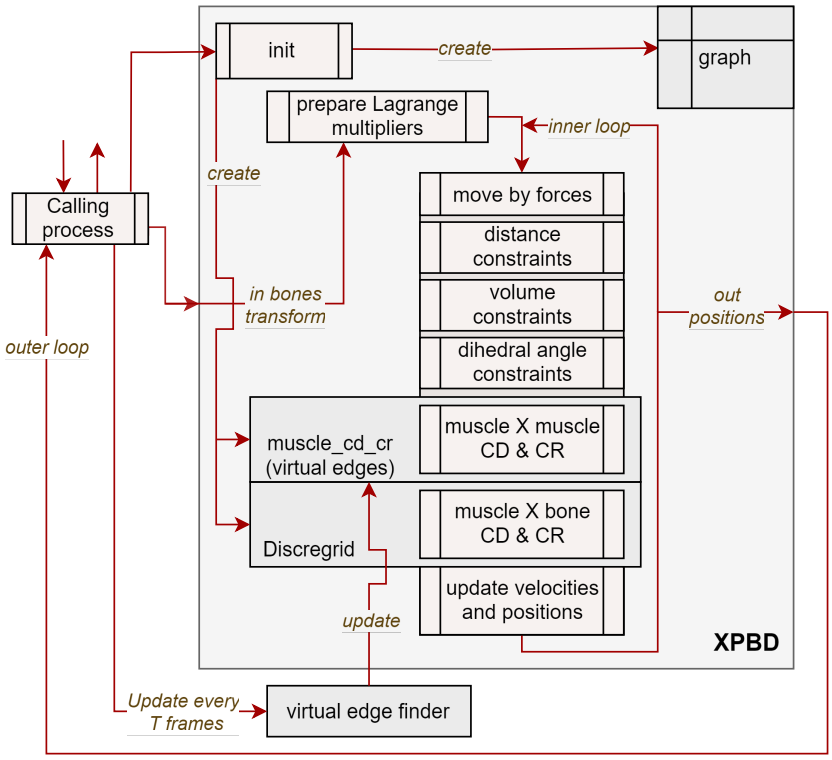


Figure 5.1: Basic control flow graph of the XPBD algorithm

This implementation should be flexible, easily maintainable and efficient. For that purpose, the modern *C++* programming language provides many tools, such as static generic classes using the templates, and the **Curiously Recurring Template Pattern** (CRTP) possibility to implement static polymorphism. The implementation should not contain all mathematical operations done from scratch, since this would probably be a difficult process, which most likely would not match the capabilities of modern mathematical libraries, such as the **Eigen** library, which accelerates the operations using, for example, the expression templates, efficient matrix multiplications, and so on. But first, the solver should also contain the method to solve for collisions of muscles to bones, as this is an integral part of a scene containing elastic as well as rigid objects.

### 5.1.1 Muscle-to-bone collisions

As the Discregrid library used in the current state of the PBD deformation algorithm works well, there is no immediate reason to choose anything else for this purpose. One possible modification is to also parallelize this process of querying the SDFs with each muscle vertex. This is possible due to the fact, that the method of the Discregrid library used to get the distance, the direction and the sign to the bounded

surface does not change the state of the underlying signed distance function object. Compared to the results shown in the PBD paper [KČ21], the results of using Discregrid for muscle-to-bone collisions can be seen in Figure 4.1 on page 49, where during the hip flexion, the muscle is no longer caught in between the *acetabulum* and the *femur* head but is instead being unnaturally bent.

### 5.1.2 Constraint design

This thesis proposes the constraint memory and execution to be done on the CPU due to the limited scope of this thesis, although a GPU implementation could provide better time efficiency, given that the individual constraint projections could be done entirely on the GPU without unnecessary copying of positions, which would be only required once at the end of the outer loop. Since matrix multiplication is needed for the constraint projection, a general-purpose-GPU parallel computing platform such as the **NVIDIA® CUDA® Toolkit** (developer.nvidia.com/cuda-toolkit) could be employed in the future.

Moreover, the CRTP for the constraint centralized functionality (mainly the Lagrange multiplier and the projection method, containing Equations 3.7 and 3.6) is proposed. Static polymorphism allows for shared functionality across instances of deriving classes while avoiding runtime look-ups in the virtual table. With modern compilers, this may seem like a premature micro-optimisation, but considering the number of constraints on detailed meshes can reach up to hundreds of thousands even in moderately populated scenes, and that these constraints are all projected e.g. 100 times per one simulation frame, the memory spatial locality of the methods and the class members becomes of great concern, which the compiler might miss during optimisation if classic inheritance with virtual methods had been used. The CRTP will be implemented as is shown for the distance constraint (Figure 5.1). All the constraints (distance, dihedral angle, volume) will hold the same definitions of the **gradient** and the **cost** function as in the original PBD paper [Mül+07].

To foreshadow, the only difference in the distance constraint formulation should be that the constraints will **not necessarily preserve the original distance**, but a desired distance as a dynamic parameter (which can also be the original one). This will be explained further in the context of muscle contraction modelling.

Each defined constraint must implement the methods for the computation of the gradient and the cost, as the projection method uses them. One drawback of this approach is that the number of particles (vertices) must be fixed, which is not problematic with the distance or dihedral angle constraints, where the number of particles corresponds to 2 and 4, respectively, but becomes problematic with e.g. the volume constraint, which may influence an arbitrary number of vertices (since the number of vertices per muscle mesh most likely varies) and the projection for it has

to be implemented separately.

Most of the constraint instances should hold a floating point number of a Lagrange multiplier. Those that would not, are desired to be infinitely stiff, without compliance, e.g. the collision detection and resolution. The volume constraint could be made a special case, as the volume should be preserved at all times unless some highly degenerated triangles emerge. Such a behaviour can be modelled by setting the stiffness of this constraint very high. For this thesis, the decision has been made not to include the Rayleigh dissipation potential for constraint damping formulation for the volume constraint, since indeed, this constraint is desired to give as little a compromise as possible, hence the conscious decision to also give it the compliance of zero was made, although the constraint works well even if it is not zero. This particular projection uses Equation 3.5 for the $\Delta\lambda$ computation.

Source code 5.1: CRTP design of constraint implementation

```
template<typename Derived, int ParticlesNumber>
class constraint {
    double m_lambda = 0;
    ⋮
    // Uses methods grad(...) and cost() of the Derived class.
    void project() {
        ... compute Δλ with Equation 3.7
        m_lambda += Δλ;
        ... compute Δx with Equation 3.6
        x += Δx;
    }
}
class distanceConstraint :
public constraint<distanceConstraint, 2> {
    double m_desired_length;
    ⋮
    void grad(double* grad) const;
    double cost() const;
}
```

### 5.1.3 Paralellization

To achieve higher time efficiency of the projection, which in turn allows for more inner solver iterations while keeping the simulation fast enough at least for the developer during testing, the projections of constraints can be parallelised using e.g. the Parallel Standard *C++* Library (PSTL, github.com/llvm-mirror/pstl) or the preprocessor directives of the OpenMP parallelization API (openmp.org/). Of course, the same parallelization technique can be applied to GPU parallel processing.

As previously noted in section 3.2.3.6, the order of constraint types projections has to stay constant to prevent oscillations. Whereas on the level of individual constraint type, the constraints of that type can be projected in a parallel fashion, given those parallel projections do not affect the same particles (vertices) [BMM17]. To do that, the constraints of each type have to be separated into edge-independent sets, as illustrated by Figure 5.2, where the independent set for distance constraints is in green (edges, that do not share a vertex), and for the dihedral angle constraints, is in blue (closed quadruples, that do not share a vertex). Note, that the illustrated colours may overlap, as the different types must be projected sequentially. The most basic way to achieve this is to use a graph-colouring technique.

The graph colouring does not proceed on the mesh itself, but on a graph, where the nodes are the individual constraints and an edge is present if the other constraint affects the same particle (vertex). An example of such a graph for distance constraints can be seen in Figure 5.3, where three sets of independent distance constraints are found. Note, that the nodes in the green overlaying graph correspond to the edges (one distance constraint per edge). The distance constraints belonging to the same colour can be processed in parallel. The different colours, though, have to be processed sequentially. Hence, the example in the figure would produce three sequential phases of parallel processing.

The situation becomes much more complex when more particles are affected

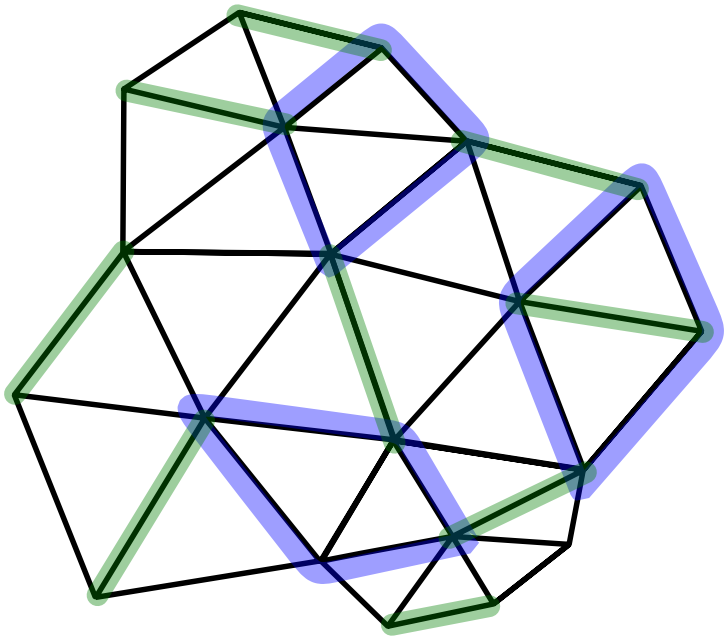

Figure 5.2: Edge-independent sets of constraints for distance (green) and dihedral angle (blue) preservation

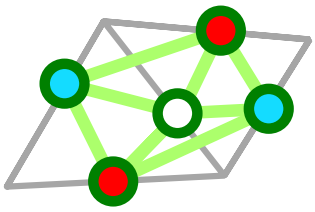

Figure 5.3: Coloured distance constraint graph overlaying a mesh

by a single constraint. In the case of the dihedral constraint, the illustration can be found in Figure 5.4. In the figure, each edge that has two adjacent triangles represents a constraint. Within a closed fan and any adjacent constraint (the red edges), the graph is complete (all the constraints share the centre vertex). In the example, this forces seven colours to be used. The first truly parallel (more than one constraint) set with the red (or yellow) node colour can be found only within the second ring of the fan (node with the dark red outgoing edges).

To perform the graph colouring itself, considering the limited scope of this thesis, **greedy colouring** with the **smallest-last ordering** (SLO) will be used, where before the start of the greedy colouring, the nodes of the graph are sorted in the descending order of their node degrees (number of outgoing edges). This greedy approach can be used, as there is no need to meet a certain upper limit of the colour counts. The results may be sub-optimal (many colours) but should speed the inner solver up anyway.

The greedy colouring algorithm keeps asserting the colours to nodes if they are available (no neighbour shares the same colour) at the current processing step. In the worst case, this may produce the same number of colours as there are nodes (no

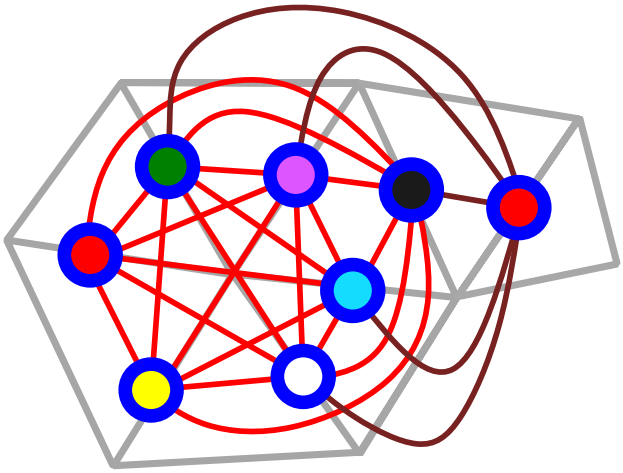

Figure 5.4: Coloured dihedral angle constraint graph overlaying a mesh

constraints would be processed in parallel).

Further, for the per-muscle mesh volume constraint, the muscle volumes are independent of each other (no volume constraint pair shares the same particles). Hence, they can be trivially projected in parallel.

Having presented the modelling environment changes, the two main proposals of this thesis will continue. The next section describe an approach to avoid muscle collisions, a form of passive intermuscular interaction.

## 5.2 Passive intermuscular interactions

Due to the need for a real-time simulator, there is also a need for time-efficient collision detection among muscles. The efficiency of the collision resolution is then directly dependent on the type and quantity of detection provided information. A proposal of a collision-solving algorithm to avoid intermuscular penetration follows.

### 5.2.1 Virtual edges

Firstly, preventing the classical collision-solving structures update, the presented collision-solving algorithm could be described as a structure-less one. An important starting point is that the muscles change in shape dynamically but only in geometry while the topology stays the same (no tearing or joining occurs). The following algorithm could be classified as a stochastic one. In stochastic collision handling approaches, the detection of just the majority of collisions suffices. The stochasticity of this proposed algorithm is not so much of a random sense but more of a hopeful (probable) collision avoidance strategy, while not explicitly defining the probability of it. The main goal is for the detection to be efficient and the collision response to resolve the unrealistic behaviour. Another reason for leniency is that even the meshes are just approximations of the real shapes, moreover, one collision response could also resolve the collisions in the neighbourhood since the muscles are elastic objects.

Secondly, exploiting the *a priori* observation, that a musculoskeletal scene contains muscles usually in an **anatomically and physiologically pre-defined relationship of closeness**, the search for collisions in an unknown scene (where for example the user could move the muscles arbitrarily) can be reduced to the search in the subspaces of predictable local contacts of the typically close muscle areas and their neighbourhoods. This observation allows the proposed approach to partially skip the broad-phase collision detection, as it is often well-predictable.

The fundamental element is the so-called virtual edge $h = (a, b)$, where the vertex index $a$ lies in one muscle vertex set $M_1$ while the vertex index $b$ lies in

another muscle vertex set $M_2$. These indexes, in theory, could be chosen **arbitrarily** from the sets. The role of this element is to scout its neighbourhood between two muscle surfaces to find the areas of so-called **Contact Distance Proximity** (CDP). CDP is the space between the two meshes, where a collision could potentially occur in the future. The virtual edges are designed to quickly and dynamically avoid them based on the assumption, that no undefined muscle behaviour disturbs the scene.

Despite anatomically and physiologically predictable collision areas, no such explicit information is provided in the musculoskeletal model. Therefore, an initial configuration resembling the *a priori* information must be found as the starting point of a set of virtual edges.

#### 5.2.1.1 Finding virtual edges configuration

There are two scenarios where the configuration of virtual edges needs to be found, particularly

1. at the beginning of the simulation
2. and during the periodic update during simulation to better reflect the updated geometries.

Either way, the goal is to find the initial configuration of all virtual edges (across all elastic objects), which will be the starting point of each virtual edge Zig-Zag descent. More specifically, the problem is to search for vertex index pairs representing virtual edges among all $\frac{N*(N-1)}{2}$ unique pairs of $N$ muscles.

A presumably good solution for one pair of muscles would be to find random initial virtual edges uniformly distributed across geometries. This seemingly trivial random initialisation is not so straightforward considering the meshes may contain many degenerated triangles making uniformity hard to obtain. Some proximity areas may be completely missed due to the randomness. The corresponding vertices of such one virtual edge may also be unreasonably too far away from each other taking too many steps to meet at a proximity area and even arguably introducing more local optimum deadlocks on the way.

Considering the naivety of the movement of the virtual edge (described in detail in the next section) let's, instead, find a more reliable initialisation and update of the virtual edge configuration method using a more classical approach of space partitioning. Such one approach could be for example using a BVH with AABBs as nodes, which are usually implemented to provide quite fast results for point-to-mesh proximity queries.

In practice, this means taking each unique pair of muscles and for all vertices of one of the muscles query the BVH built over the other muscle for great enough closeness proximity. The BVH structure is usually capable of also returning the index

of the closest vertex in the other muscle within that closeness radius. The closeness radius is usually a parameter to the query and often helps with search acceleration. This parameter is then the upper bound of the initial virtual edge lengths and should be chosen at least bigger than what is the CPD.

Having a way to find one virtual edges configuration it is then possible to update the virtual edges periodically in each $T$ frame of the simulation, while during the frame solver iterations the virtual edges

1. reset each iteration to the initial configuration
2. or stay where the last iteration left them.

On one hand, the first case limits the virtual edges to the proximity of the muscles and their near neighbourhood at the time of configuration update. On the other hand, the second approach allows the virtual edges to travel greater distances, potentially detecting *unexpected* collisions.

In between the configuration updates, each virtual edge follows the following **Zig-Zag algorithm** to scout its neighbourhood for collisions.

#### 5.2.1.2 Zig-Zag algorithm

The first step is to consider index $b$ in a $M_2$ subset with all its neighbouring vertex indexes $\delta(b)$ of the size $s = |\delta(b)|$, hence considering vertex indices $\{b \cup \delta(b)\}$. For each of them, calculate the distance $d^a_{0,1,2,\dots,s,s+1}$ to the reference vertex index $a$. From these distances, pick the minimum and the corresponding vertex index. Change the index $b$ to this minimising neighbour $b = \arg\min_{i \in \{0,1,2,\dots,s,s+1\}} d^a_i$ (or let it stay in case it is the minimal one).

The second step is the same as the first one, only this time it is vertex index $a$ which might get changed to its minimising neighbour in respect to the reference vertex with the index $b$. The first step is visualised by the image 5.5. Vertex index $b$ is swapped for its neighbour who minimises the virtual edge length. The next step probably swaps the vertex index $a$ also.

Since the distance serves only for relative comparison it is not needed for it to have a physical correspondence, therefore arbitrary squared Euclidean distance between vertices was chosen. This distance $d^p_q$ between two vertices $p$ and $q$ is given by Equation 5.1 and is chosen as the computationally feasible variant of the classical Euclidean distance, where the computation of the squared root is needed.

$$d^p_q = \sum_{i=1}^{n} (q_i - p_i)^2 \tag{5.1}$$

This can be regarded as a gradient descent method where every time a vertex index of the virtual edge is changed, the objective function changes (because it is the

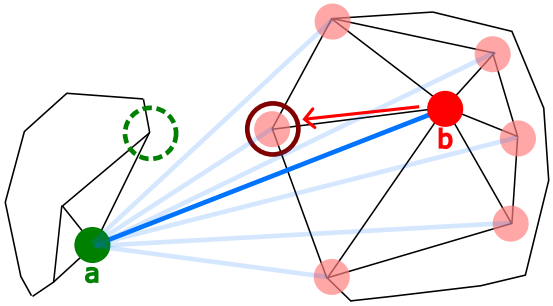


Figure 5.5: The first step of the Zig-Zag virtual edge algorithm

distance to the referential vertex). This Zig-Zag step is repeated in many iterations. The convergence criteria (depicted by Figure 5.6) are either

1. reaching the maximum iteration count `max_iter`,
2. local minimum stagnation (none of the two Zig-Zag steps change the virtual edge and the geometry does not change until the next solver iteration)
3. or the sufficiently small length of the virtual edge is reached $d(h) < threshold$ where a collision can be expected and prevented. This user-defined threshold is defined as the **CPD**.

Figure 5.6 depicts the first step of the Zig-Zag algorithm of the vertex index $a$ movement and the possible ending states of that movement. On the left, the vertex gets caught up in a local minima while not being able to recover from it until the neighbouring geometry changes. In the middle of the figure, the vertex with index $a$ converges up to the proximity of the vertex with index $b$. The last illustration in Figure 5.6 depicts a scenario where the moving vertex of the virtual edge stops due to it reaching maximal iterations.

If many such virtual edges undergo the Zig-Zag iterations concurrently for the same geometry, probably, some of them (with the presumption of enough iterations) converge to the vertices which are close to the contact distance between the two muscles, in other words, their distance is less than CPD. This is where the collision is to be expected, hence this method could be classified as a priori collision detection method.

It is also presumable, that many of the virtual edges lay stagnant in the local minima (especially with the complex muscle geometries).

There is no less chance that a collision will be missed. Imagine the geometry changes rapidly in a short time. By the time the virtual edges converge to the new minima if at all, the muscle intersection could be so massive the collision would become impossible to repair. This is why it is also needed to periodically update the

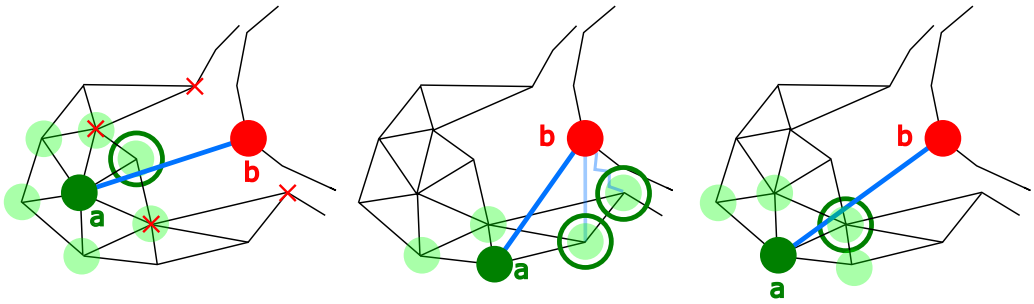


Figure 5.6: Possible endings of virtual edge movement

initial configuration of the virtual edges. It is possible to randomly add new virtual edges or use some structure-aware more precise algorithm using for example space partitioning.

#### 5.2.1.3 Degenerated edges

There are three initially distinct virtual edges $h_i = (a_i, b); \quad i = 1, 2, 3$ illustrated in Figure 5.7 which all degenerate into just one virtual edge $h_{new} = (a_{new}, b)$. It is the case that these degenerated virtual edges are not going to escape degeneracy (representing just one virtual edge while being computed three times) until the next virtual edge configuration update even if the geometry changes. At most, this particular virtual edge would be provided thrice more Zig-Zag iterations in case of sequential processing. One solution would be to detect this phenomenon and remove the redundant edges completely, another option could involve letting this degenerated edge enjoy more Zig-Zag iterations while synchronicity is provided or we might consider taking advantage of this problem to accelerate collision detection using so-called wormholes.

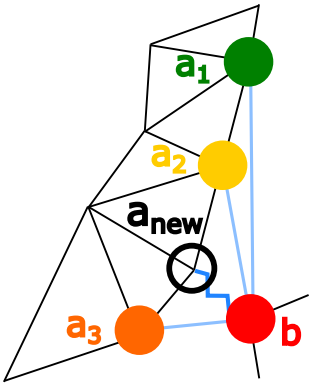


Figure 5.7: Three virtual edges about to become degenerated into one

### 5.2.1.4 Exploiting degenerated edges as wormholes

Consider sequential processing of virtual edges Zig-Zag iterations as in one virtual edge ends in one of the three states Figure 5.6 shows and then the next one is processed and so on. Such sequential processing of two virtual edges is described in Figure 5.8. If the first one to get processed would be the edge $h_2$ and then the edge $h_1$ which due to chance ends up in the starting position of $h_2$ after the first step of Zig-Zag it is obvious that $h_1$ would end up in the position $h_2'$ if it had enough iterations, even potentially degenerating there. What could accelerate the space search of the virtual edges as a whole would then be to teleport edge $h_1$ through the whole path of $h_2$ and let it continue with its iterations further and hopefully detect the dangerously close vertices $a$ and $b$. On the other hand, in case the edge $h_2'$ is in local minimum this warp would degenerate $h_1$ with $h_2$. Furthermore, if the edge $h_2'$ would signal detection of contact proximity and this contact would be resolved as a part of $h_2$ processing, the warp would provide a chance to detect more close encounters of the two muscles.

Moreover, cases could emerge where letting the edge $h_1$ end upon reaching maximal iteration count somewhere on the path of $h_2$ would be beneficial since it is probable that in the next simulation step, the geometry changes and these two virtual edges would then diverge from each other, providing better distribution of these contact detectors.

Due to the need for synchronisation, this exploitation would be unfeasible to parallelise. Various barriers for read/write global memory operations would be needed since the virtual edges can virtually wind up in arbitrary positions. In other words,

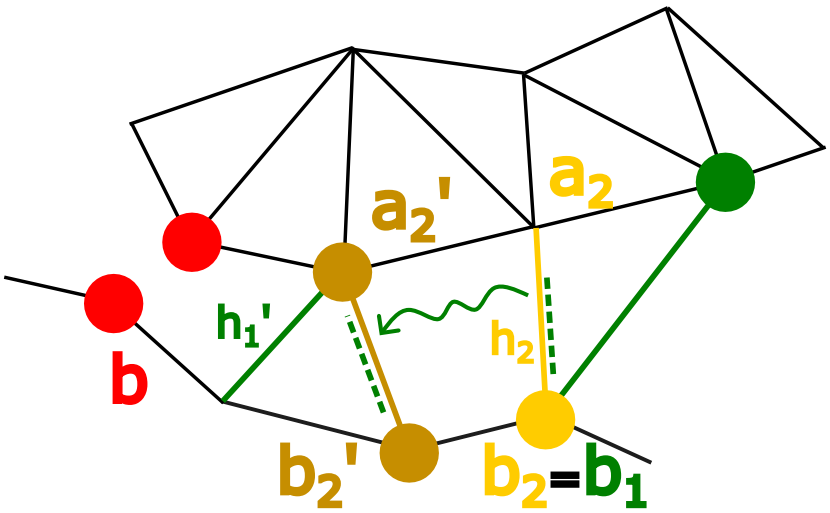


Figure 5.8: Wormhole warping of one virtual edge

many virtual edges from all over the meshes can gather at one specific area, making them all dependent on each other during such synchronous wormhole warping operations.

Even without using the wormholes, when the collisions are **sequentially avoided**, the next virtual edges incoming to the already resolved position operate on different geometry, providing **more opportunities** to find the next near-collision. In this scenario, these degenerated edges would not be entirely *degenerated*, but rather **useful**.

### 5.2.2 Virtual edge collision resolution

When the distance between two virtual edges becomes shorter than the CPD set by the user, it indicates that there could be a collision between the vertices. To prevent this collision, a straightforward solution is to only consider the vertices themselves and ignore their topology or geometry neighbourhood. By pushing these vertices away from each other, a safe distance can be created between them, effectively avoiding the collision (which may not have happened in the first place).

Such trivial resolution is illustrated in Figure 5.9, where the virtual edge $h = (a, b)$ becomes so short it may indicate an upcoming collision ($d(h) < CPD$). The resolution is to push the vertex with index $a$ by the vector $\overrightarrow{ba}$ and the vertex with index $b$ by the $\overrightarrow{ab}$ vector, each by the distance $\sqrt{\frac{1}{d(h)}} * CPD * 0.5$, where the first term norms the vector, the second term multiplies it by (at least) the desired safe distance and the last term distributes the shift among the two vertices.

The immediate downfall of this resolution is if the CPD is set too small and the external forces acting on the particles represented by these vertices are greater than CPD, the externally pushed vertex penetrates the surface as is illustrated in Figure 5.10. Starting with the vertex at position 1, the external force $ext_1$ acts on it

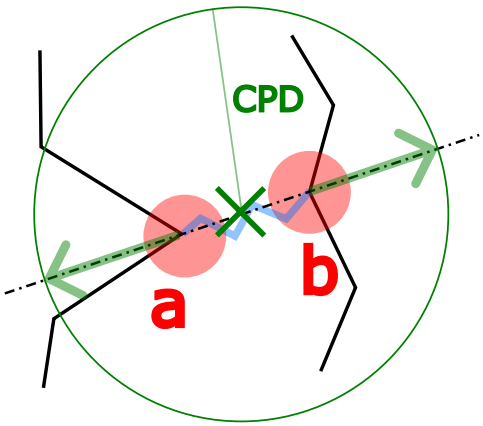


Figure 5.9: Trivial virtual edge collision resolution

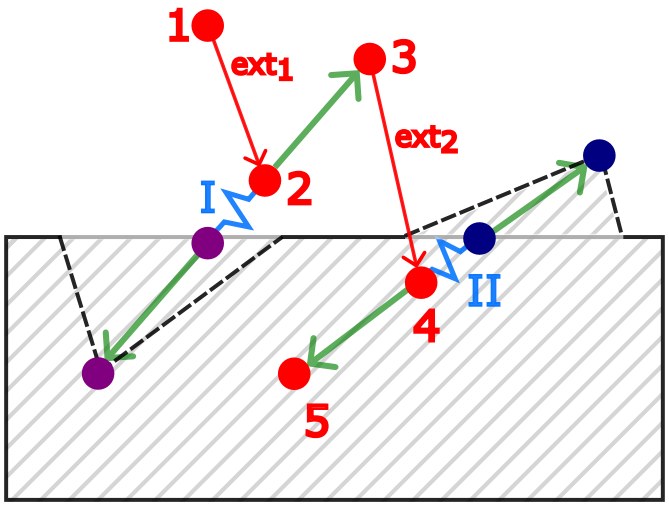


Figure 5.10: Trivial solution fails to prevent collision among muscles

and pushes it to position 2. Collision I with the purple vertex on the muscle surface is detected and successfully resolved by pushing the vertex away from the surface into position 3. There, however, another external force $ext_2$ acts in the next iteration of the solver and the vertex is pushed under the muscle surface (4). Trivial solution II incorrectly pushes the vertex even deeper into the muscle (5), and the blue vertex on the surface of the muscle, in turn, is incorrectly pushed outwards.

Adaptive CPD could be employed to reflect the magnitudes of external forces better to prevent this failure of resolution. Another idea is to develop a resolution algorithm reflective of the vertex's previous position. This idea is sketched out in Figure 5.11.

In Figure 5.11, the updated positions of the virtual edge $h = (a', b' = b)$ are examined following the detection of a possible collision. Vertex $a'$ has undergone significant changes in its spatial relationship to $b$ in comparison to its prior position $a$ (which can be quantified by the angle between the three positions). Consequently, it is probable that the boundary of muscle $M_2$ may have been breached. Correcting the position to $a^{c1}$ would not resolve the collision. A more effective approach entails computing the vector $\overrightarrow{v} = \overrightarrow{Sa^{c1}}$ and the vector $\overrightarrow{a} = \overrightarrow{a'a}$, followed by either summing the normalized vectors using Equation 5.2 to obtain $\overrightarrow{c2}$, or summing and normalizing using Equation 5.3 to obtain $\overrightarrow{c3}$. The resulting position $a^{ci}$, where $i = 2$ or 3, is then expressed as Equation 5.4. To preserve the CPD, it becomes imperative to relocate the opposite vertex $b$ towards the position $b^{ci}$, where $i = 2$ or 3, or compute the same correction for this vertex and average the two resulting directions (with one of them negated) to obtain an average direction the new positions must travel (one of them in the negative direction) to meet the boundary of the circle and preserve

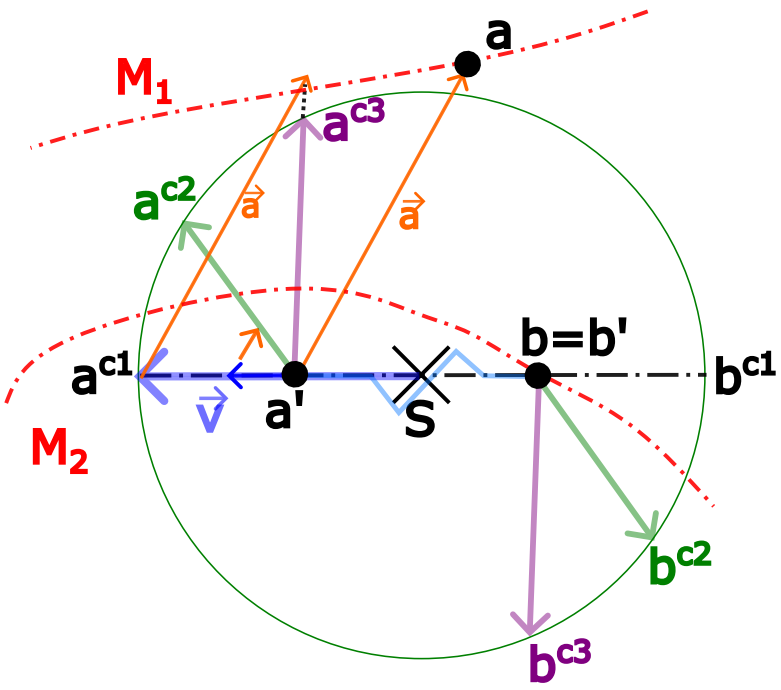


Figure 5.11: Previous position reflective collision solving

the CPD. Both variations of the more intricate correction method would effectively resolve the collision in this instance, surpassing the simplistic solution.

$$\overrightarrow{c2} = \frac{\overrightarrow{v}}{|\overrightarrow{v}|} + \frac{\overrightarrow{a}}{|\overrightarrow{a}|} \tag{5.2}$$

$$\overrightarrow{c3} = \frac{\overrightarrow{v} + \overrightarrow{a}}{|\overrightarrow{v} + \overrightarrow{a}|} \tag{5.3}$$

$$a^{ci} = \frac{a' + \overrightarrow{ci}}{|a' + \overrightarrow{ci}|} \times CPD \tag{5.4}$$

## 5.3 Active intermuscular interaction

The last proposal on top of the creation of an in-theory-advanced XPBD model compared to the former PBD and the passive intermuscular interaction, is the facilitation of active muscle interaction in the form of synchronised, contractions dependent on the current orientation of the underlying joint. In this context, the terms muscle *activation* and muscle *contraction* will be used interchangeably.

Although a motoneuron model, where the activations of muscles could be gathered using the EMG, would probably be the most realistic to simulate the interacting contractions of the muscles (or even just the contracting parts of them), a simplified

model of the muscle activations patterns can be used, as this proposition is more a proof of concept than a final product.

This is why, as of now, the model of muscular interaction will be realised, in this context of inverse kinematics, as a **joint-orientation-governed orchestration of the muscles** (that would, in reality, lead the movement) to simulate their **activations** through the use of **dynamic XPBD distance constraints** (with varying parameters).

### 5.3.1 Movement-inducing joint representation

Although a physiological joint does not move, rather the bone which is situated inside of it, in the context of modelling the active muscular interactions, a joint has been chosen to represent the centre of control for the muscles around it, moving the bone the joint contains. This is not supposed to be a physiological analogy, but rather a conceptualisation of the centralised control over the active muscle interactions, where the muscles surrounding the joint interact primarily to move the bone, but also to stabilise the joint and its surroundings (in the case of the hip joint, the *pelvis* bone is often incorporated into the movement of the *femur* as the *pelvis* represents the non-moving base of the joint, which needs to be stabilised and the *femur* represents the bone movable relatively to the *pelvis*, which needs to be moved). The joint is not the source of the movement, nor the moved object, but rather the space, **where the movement occurs**. From this point on, such a control centre will be denoted as the *Joint Control Unit* (JCU).

This JCU (as a centre of control of the active muscle interaction), to fulfil its function of governing the surrounding muscles under a specific movement of the bone (e.g. hip flexion, where the *femur* changes its orientation in one of its three degrees of motion) must

1. be aware of **which muscles** should cause the movement of the bone,
2. know the **current orientation of the bone**,
3. own for each concerned muscle a set of **dynamic distance constraints** corresponding to **detailed surface fibres** of this muscle to be able to contract it,
4. and know **the information about all concerned muscle activations under various orientations** (samples based upon which to decide what activations to impose on the muscles it governs under a specific orientation of the bone).

The first requirement is easily obtainable either from literature [KOA19], from the specifications of the prime and the most powerful synergistic movers or, as an al-

ternative with the focus on the hip area, one of the many well-documented models of the lower extremity, such as the one presented in the paper [Raj+16] by Rajagopal et al. and later modified by Uhlrich et al. [Uhl+22] (available at simtk.org/projects/fbmod-passivecal) which had been successfully used to perform a muscle-driven simulation of the gait movement. The results of this simulation corresponded qualitatively to important characteristics of EMG data [Raj+16], and therefore, it can be deemed physiologically accurate enough for this work.

Secondly, the current orientation is implicitly available in the **Muscle Wrapping 2.0** project on the input. The only hurdle to overcome is the injection of it into the XPBD deformation algorithm, containing the JCUs.

Thirdly, to obtain the detailed surface fibres needed for contracting the muscle surface, the given option to also generate inner muscle fibres for each muscle of the original approach [KČ21] can be utilised. These inner muscle fibres can be used to estimate the surface fibres by snapping each point of the fibre polyline to the nearest surface mesh point. As the resolution of these fibres is usually lower than the number of edges when traversing the surface along the fibre, more detailed paths connecting these sparse surface fibre positions must be found to ensure a smooth muscle contraction. These detailed surface fibres are sufficient to find only at the start of the deformation, as in this state, the muscles should be in their original shape [KČ21], hopefully reflecting the anatomy with the best precision.

In regards to the fourth (and last) point of requirements (to gather samples about muscle activations under various bone orientations), a tool to estimate muscle activations called **the static optimization** during arbitrary movements is available in the **OpenSim** (simtk.org/projects/opensim). Used on the detailed model Rajagopal et al. created [Raj+16] and Uhlrich et al. modified [Uhl+22], this tool should provide sufficient muscle activations as a good starting point for this proposed proof of concept (given the right tuning of the tool). Further explanation of points **3.** and **4.** follows.

#### 5.3.1.1 Detailed surface fibres extraction

Given the generated inner surface fibres by e.g. the Kohout & Kukačka [KK14] algorithm, a surface representation of them should be made, because the XPBD operates on a surface mesh in this case. An alternative would be to select a fibre running through the centre of the volume (or an average fibre) and to use that for contractions while parametrising the positions of the surrounding vertices as can be done using Mesh Skinning described 3.2.3.2 on the page 28. Nevertheless, a pure surface-based representation has been chosen due to its simplicity. Here, the assumption is made that the resolution of the inner fibres is less than that of the mesh.

For each of the inner fibre polyline vertices, the closest point on the muscle

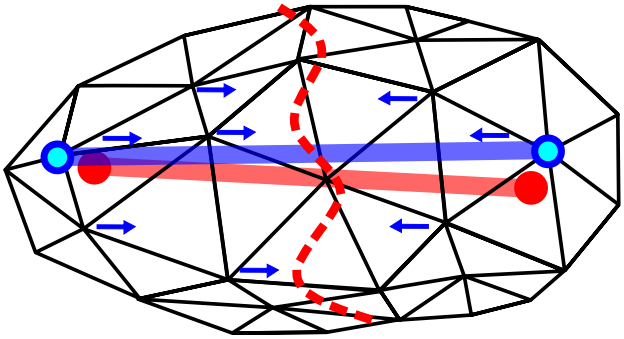

Figure 5.12: Crease formation due to sparse surface fibres contraction

surface is found. The connectivity of the fibres stays the same. In case the dynamic distance constraints would be made for each of these sparse polyline edges and these constraints would contract (try to minimise the distance of the two concerned vertices), a crease would start to form between these two vertices, as illustrated in Figure 5.12 by the dashed, red curve. In the example, the inner fibre is illustrated in red, the surface fibre is illustrated in blue, and the blue arrows represent the propagated forces causing the crease. This is because of the propagation of forces during XPBD deformation.

To overcome this behaviour, every inner fibre segment can be approximated by a polyline, whose points form a minimal distance path between the original surface points on the mesh, as Figure 5.13 depicts using the same colours. To find such a path between two points on a mesh, the **Dijsktra** algorithm can be utilised. With these detailed surface fibre segments, the shortening or lengthening of the constrained edges would most probably happen only locally, while not disturbing the overall shape of the muscle in the majority of the cases. The cases, where even this approach would fail to produce adequate dynamic constraints, can happen if e.g. the mesh contains degenerate triangles.

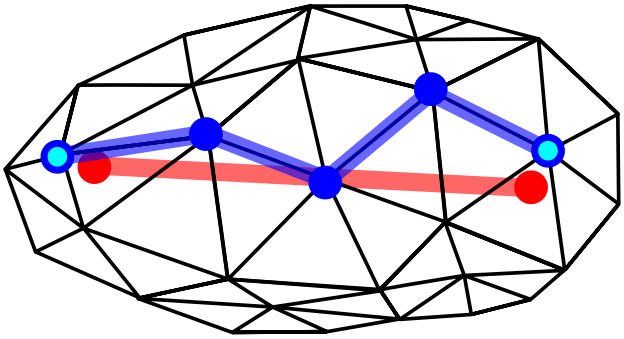

Figure 5.13: Detailed surface fibre approximating the sparse inner fibre

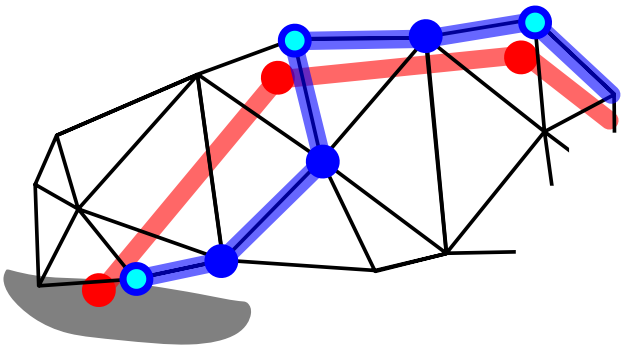

Figure 5.14: Shortest distance path producing perpendicular surface segments

Another ill-posed case for this method of generating detailed surface fibres is when the two vertices of an inner fibre segment find their closest mesh vertices on the opposite sides of the mesh. This situation arises typically near the attachment area of the muscle or near thin muscle areas. A 2D example of this situation is depicted in Figure 5.14, where the leftmost red inner fibre vertex is attached to the grey bone but continues to the right on the other side of the muscle. The resulting detailed surface fibres for this first inner segment are forces to wrap around the muscle, possibly running perpendicular to the original inner fibre direction. A naive solution would be to generate the inner fibres with higher resolution or count in hopes that they would follow the surface more closely.

It should be noted, that these surface fibres are not supposed to represent the physiological fibres, which, as discussed in section 2.1.4 on the page 11, are not always running in parallel. Rather, these fibres represent the estimations of directions of forces the muscle can generate.

#### 5.3.1.2 JCU bone orientation sampling

For a single JCU governing surrounding muscles, samples of all concerned muscle activations under various orientations (movements) should be made. These samples will later serve to infer the activations for an arbitrary hip movement using direct correspondence to the sampled activations or interpolation of the orientation.

If considering the example of the *femur*, it has three DOFs. It can flex or extend, abduct or adduct, and internally or externally rotate. Therefore, any orientation of the hip joint can be expressed by three scalar values, representing the degree of rotation around each of the axes. This is the space where the samples should be collected (Figure 5.15).

This space is theoretically limited by the ranges of motion, but in reality, the bounds are not defined by a rectangular prism (as the figure shows), but rather by a deformed ellipsoid, accounting for muscle imbalances or injuries.

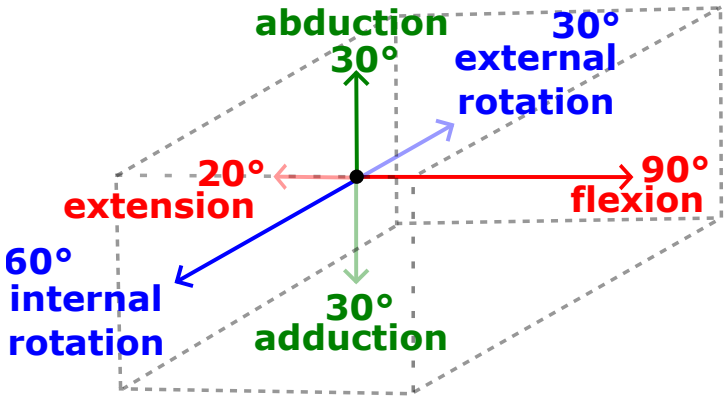


Figure 5.15: The parametric space of a JCU's bone's orientations limited by theoretical ranges of motion

#### 5.3.1.3 Activation estimations

As previously described, the OpenSim software provides a tool for estimating muscle activations during movements called the **Static Optimization Tool**. Various explanations about the tool can be found in

a. the OpenSim Frequently Asked Questions (FAQs) on the web page opensim.stanford.edu/downloads/WebinarMarch2011FAQ.pdf,

b. the OpenSim Documentation available at opensimconfluence.atlassian.net under

   1. User's Guide → Static Optimization,
   2. Examples and Tutorials → Intermediate Examples → → Working with Static Optimization,

c. the complementary lecture videos for the book Biomechanics of Movement [UD21] by Uchida & Delp,

d. and the article done by Andreson & Pandy [AP01], which also compares it with a dynamic optimization approach.

The prerequisite to static optimization is the computation of net forces and moments in each joint. In reality, the joint is moved by many complex biological structures, such as the muscles or ligaments. This is modelled using an equivalent (but simplified) representation of net (sum of individual) joint forces and net joint moments. The joint forces correspond to when the muscle contracts and the joint moments correspond to the forces causing joint rotation [UD21].

To compute the net joint moments and forces, an **inverse dynamics** model is used. For each segment of the motion (e.g. each bone), motion equations are formed

for the positions of end vertices, velocities and accelerations (the latter two by the differentiation of the positions). Then Newton's second law is used to compute the moments and forces for this segment concerning the centre of mass of the segment (to simplify the moment computation) [UD21].

Now, knowing the net force and net moment for a particular joint, analogously to when the body has to decide which muscles to active for this resulting force and moment, so has the model. This problem may be called the problem of **muscle force distribution** [UD21].

A net ankle moment (in flexion/extension range of motion) distribution will be used as an example. In the problem, the net moment **M** is equal to the sum of the moments of flexion across all major *flexors* minus the sum of all extension moments across all major *extensors* (Equation 5.5) [UD21].

$$\mathbf{M} = \sum_{f=1}^{nf} \mathbf{F}_f \mathbf{r}_f - \sum_{e=1}^{ne} \mathbf{F}_e \mathbf{r}_e \tag{5.5}$$

Where **nf** and **ne** denote the number of *flexors* and *extensors*, respectively, $\mathbf{F}_f$ is the force of the *flexor* $f$, $\mathbf{r}_f$ is the moment arm of the *flexor* $f$, and the force and moment arm for *extensor* $e$ is defined analogously. Given one equation, where the $M$ is a known variable and the rest ($nf + ne$) are unknown variables, the system is underdetermined [UD21].

A naive way of finding the solution would be to reduce the number of unknowns by preventing the corresponding muscles from acting or making their contributions constant. Another solution would be to add equations assuming a group of muscles would generate the same force. Generally, an optimization formulation is needed, where an objective function is minimalised while satisfying inequality, equality, and boundary constraints (similar to the XPBD method). The boundary constraints for this system would be the limits of forces every muscle can generate [UD21].

The objective function present in the OpenSim **Static Optimization Tool** (SOT) (opensimconfluence.atlassian.net/wiki/spaces/OpenSim/pages/53089619/How+Static-+Optimization+Works) is defined as Equation 5.6. Minimalisation of this function would mean the minimalisation of overall muscle activations $\mathbf{a}_m$, raised to the user-defined exponent $p$. The objective function can also represent the total muscle stress, instead of total muscle activations, as mentioned in the lectures by Thomas K. Uchida, Ph.D., P.Eng. [UD21], which could provide better results, but the official documentation of the OpenSim describes the objective function in this form, which may relate to the muscle stress in the end, as will be later discussed.

$$\mathbf{J} = \sum_{m=1}^{n} (\mathbf{a}_m)^p \tag{5.6}$$

For the minimalisation of the objection function 5.6, the Equation 5.5 now serves as one of the constraints on the system [UD21]. In a general case, over many joints, it can be re-written while incorporating the limit force constraints as Equation 5.8 (provided in the OpenSim SOT documentation) representing the constraint relative the (generalized) net force $\tau_j$ to the joint axis $j$, where $n$ is the number of concerned muscles, $\mathbf{a}_m$ is the activation of muscle $m$, $\mathbf{F}_m^0$ is the maximum isometric force muscle $m$ can produce, and $\mathbf{r}_{m,j}$ is the moment arm of muscle $m$, as before, only this time specified to the $j^{th}$ joint axis. In this formulation of the constraint, each muscle would play the role of a **ideal force generator** [AP01].

The correspondence between Equation 5.5 and 5.8 is explained by Equation 5.7 [AP01]. Andreson & Pandy note, that in this ideal force generator case, arguably not correspondent to reality, the muscle activation $\mathbf{a}_m$ of muscle $m$ is in fact "equal to muscle stress multiplied by some proportionality constant" [AP01]. In the Equation 5.7, $k$ denotes this proportionality constant, while $PCSA$ denotes the physiological cross-sectional area, making the relation $\frac{\mathbf{F}_m}{PCSA}$ express muscular stress [AP01].

$$\mathbf{a}_m = \frac{\mathbf{F}_m}{\mathbf{F}_m^0} = k\frac{\mathbf{F}_m}{PCSA} \tag{5.7}$$

$$\tau_j = \sum_{m=1}^{n}(\mathbf{a}_m\mathbf{F}_m^0)\mathbf{r}_{m,j} \tag{5.8}$$

$$\tau_j = \sum_{m=1}^{n}[\mathbf{a}_m\mathbf{f}(\mathbf{F}_m^0, \mathbf{l}_m, \mathbf{v}_m)]\mathbf{r}_{m,j} \tag{5.9}$$

An alternative way of constricting the muscles during optimization is to incorporate the force-length-velocity properties. Equation 5.9 describes this alternative constraint, where $\mathbf{l}_m$ is the length of muscle $m$, $\mathbf{v}_m$ is its shortening velocity, and the function $f(\mathbf{F}_m^0, \mathbf{l}_m, \mathbf{v}_m)$ represents the force-length-velocity surface for this muscle [AP01]. In the context of this thesis, this alternative constraint can be viewed as more physiologically plausible [AP01].

The last thing the SOT needs to estimate the muscle activities during movement is the **actuators**. Each muscle represents an actuator. An actuator in the context of muscular activation can be interpreted as the converter of action potential to the mechanical contraction of the *sarcomeres* (a process described in section 2.1 on page 7). In the OpenSim SOT, the actuators serve the same purpose, without them specified, the muscles can not exert enough force for the tool to provide plausible results, as is discussed in the intermediate example at the https://opensimconfluence.atlassian.net/wiki/spaces/OpenSim/pages/53085189/Working+with+Static+Optimization web page. The documentation also discusses the **residual actuators**, which should have little impact on the estimation. Further, a modification is suggested that the motion

kinematics (bone and muscle positions during movement) are filtered at a **6 Hz cut-off frequency** before the SOT run to prevent noise in the results.

The user interface panel of the tool is depicted in Figures 5.16 and 5.17. As can be seen in the *Main Settings* panel, for the loaded model, the tool should have a desired motion loaded (e.g. from a file), and the option to filter the input coordinates is available. In the *Objective Function* setting, there is the possibility to change the exponent $p$ value (Equation 5.6) from the default value of **2**. Under the exponent parameter, there is a check-box with the option to **Use muscle force-length-velocity relation** which if left checked, uses the constraint 5.9, otherwise the 5.8 constraint will be used. On the second **Actuators and External Loads** panel (Figure 5.17), under the **Actuators** setting, an **Additional force set files** parameter should be supplied with the file containing actuator settings. These actuators should be **Appended to model's force set**, as the other option would overwrite all other actuators present in the model (e.g. the tendon ones). **External Loads** can be added in case of the desire to include e.g. gravity or ground resistance.

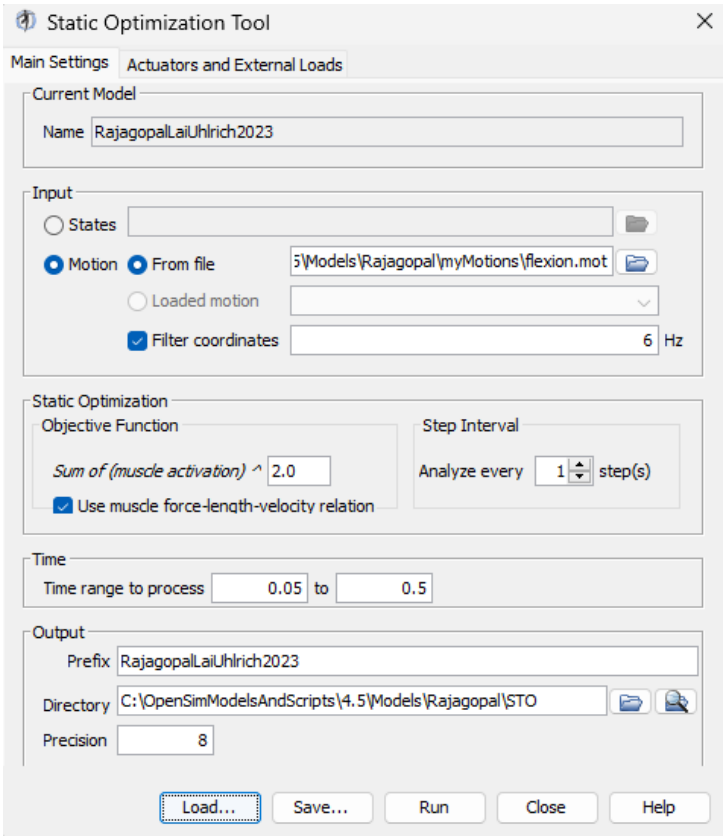


Figure 5.16: The graphical user interface of the SOT main panel

### 5.3.2 Arbitrary movement activations inference

If every sampled JCU bone orientation has an arbitrarily big vector of the magnitudes of muscle activations, and the activations are desired to be known at an arbitrary orientation in the 3D space, often, if this arbitrary orientation is not one of the sampled ones, the activations must be guessed based on the given orientation. If the samples cover enough orientations, interpolation can be used. The three DOFs of the JCU bone define a parametric space for this interpolation (Figure 5.15).

Since the OpenSim SOT uses inverse dynamics to compute the net joint forces and moments to estimate the muscle activations between individual steps of the movements, the quality and physiological correspondence of the resulting activations for an arbitrary orientation most likely depends (apart from the SOT configuration) on

- the resolution of the sampled movements (the amount of orientation change between two consecutive motion steps),
- and the order of the sampled movements (as the next sample values may depend on the current sample values).

As an example, consider the samples in Figure 5.18. The figure depicts possible sample orientations with grey spheres. Each sphere has an arbitrary dimensional vector (the number of concerned muscles) of scalar values representing the activation for each muscle. On the left, the illustration shows the axes of the parametric space, extending to the limits of ranges of motion, completely covered in the samples. These samples are chosen as the first choice since they represent all the *pure* motions (no influence of other rotations) [KOA19]. From this point on, on the right of the figure, the next sampling could, for example, continue cross-like from the maximal flexion. As the figure becomes cluttered, it is up to the viewer to also imagine the

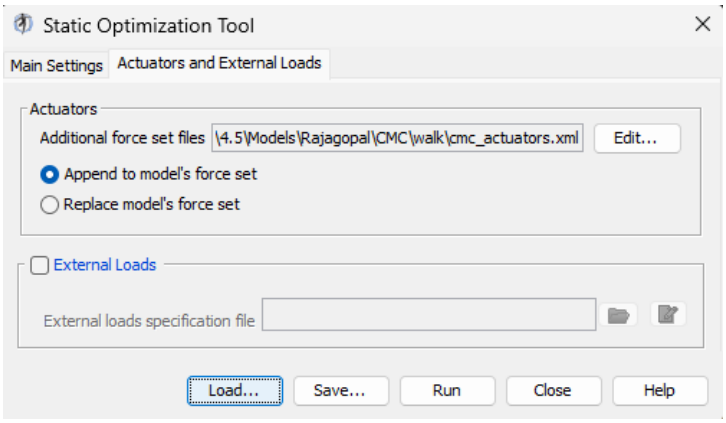


Figure 5.17: The graphical user interface of the SOT secondary panel

crosses growing from each of the remaining limit orientations shown in respective axis colours.

If such crosses would also spur from each of the pure motion orientations, this space would become uniformly sampled (in the nodes of a 3D uniform grid). In that case, the interpolation of activations for an arbitrary rotation could be done by e.g. basic bilinear interpolation, or even using the nearest-neighbour method, if the grid was dense enough. But in this thesis, the uniform grid will not be used, because

1. the order of the samples may hinder physiological correspondence of the activations,

2. and the acquisition of the samples is a lengthy process.

Instead, the idea is that only the most typical and physiologically relevant movements should be sampled to provide the most accurate results. Perhaps, in future work, these typical movements could be identified, most likely resulting in smoothly transitioning curves of samples in the orientation parametric space.

#### 5.3.2.1 Nearest neighbour search

The chosen distribution of the activation samples is sparse and non-uniform in the parametric space. Since for example, the activations during maximal extension should not influence the activations near maximal flexion, only a **subset of the samples will be used for the interpolation**. Moreover, interpolating across the whole space could become computationally unfeasible, since this interpolation is done in each of the outer XPBD solver iterations (when the orientations of the JCU bones may change).

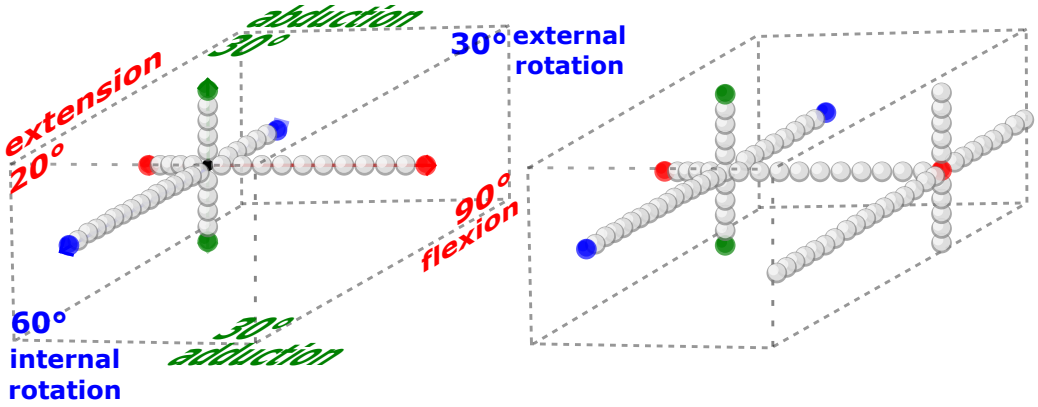


Figure 5.18: Pure motion sampling on the left, cross extension for maximal flexion on the right

For an arbitrary orientation, an influencing subset of samples must be found in an adequate time. This problem can be solved using the **k-nearest neighbours algorithm**. To accelerate the algorithm, a **kd-tree** (in this case, $k = 3$) structure is proposed due to its efficiency and ease of understanding. The structure has to be created only once, as the samples do not change during the simulation. Since the simulated motions are usually advancing in small JCU bone orientation changes, a better approach could be utilised to take advantage of this. But in the context of this work, the kd-tree suffices, as the only operation being done during simulation is finding k-nearest neighbours, where a balanced kd-tree provides the search in $\Theta(k\ log(n))$ time complexity, where the $n$ is the number of samples, which should currently be only up to a few hundred.

This structure introduces one new parameter **K** representing the number of nearest neighbours which should be found to interpolate among. This parameter should be chosen based on the density of the samples in the parametric space. Figure **??** illustrates the result of a search for the 10 nearest pink neighbours of a query orientation in black.

### 5.3.2.2 Radial basis functions for activation interpolation

Having k-nearest neighbours to the queried bone orientation, their values must be interpolated. **Radial basis functions** (RBF) are proposed to perform the interpolation since the number of neighbours may vary and may also be of an arbitrary magnitude, depending on the density of the samples. Another reason for this type of interpolation is that the further the neighbour is, the less impact it should have on the interpolated value. Narrowing the interpolation space to the neighbour subspace also raises the feasibility of this method, considering the required evaluation of an exponential function per neighbour per outer XPBD iteration.

Each neighbour represents a **Gaussian kernel** which is proposed arbitrarily as the default one. Each of these neighbour kernels $n$ gets a weight $\mathbf{w}_n$ asserted to them based on a basic exponential function 5.10, where **F** (otherwise usually denoted as $\lambda$) refers to the fall-off rate of the exponential and $\mathbf{d}^2_{q,n}$ represents the Euclidean distance between the parameter space orientations $q$ to the query orientation (otherwise usually denoted as **r** for radius) and $n$ for the neighbour orientation.

$$\mathbf{w}_n = e^{-\mathbf{F}\mathbf{d}^2_{q,n}} \tag{5.10}$$

These weights are then normalised and used for interpolation of the corresponding neighbour muscle activations (for each muscle activation, a weighted sum of all neighbours). In the case, the distance to any of the neighbours $\mathbf{d}^2_{q,n}$ smaller, than a sample difference $\epsilon$ between two floating-type scalars, the activations across muscles, for that particular neighbour, are returned directly with the weight of one, as this query is nearly identical to one of the sampled orientations.

#### 5.3.2.3 The contraction

Under a specific JCU bone orientation, having activation levels (contraction magnitude) for each of the muscles, the JCU modifies the desired lengths of the distance constraints representing detailed surface fibres of corresponding muscles.

Each constraint is analogous to a *sarcomere* in that when the whole fibre contracts, the whole muscle contracts, as described with *sarcomeres* in section 2.1 on page 7. As the activation levels on the output of the OpenSim SOT come from the $[0; 1]$ range, then a mapping must be defined to change the original distance of the *contraction* distance constraints, since for example, if the original length was simply multiplied by the complement of one of this activation level, then a contraction level of one would completely shrink the approximating *sarcomere* to a single point.

As [RMT16] notes, on average, a sarcomere contracts by $30 - 50\%$ of its resting length (resulting in $50 - 70\%$ of its resting length). Even though this change of length probably varies a lot across the body's different muscles, to reach the goal of *not letting the mesh collapse to a point*, the lower bound of this very rough estimate can be used. Therefore, the mapping describing how much a contracting distance constraint belonging to a specific muscle should change its length in the current simulation step is given by the scale 5.11, where $\mathbf{L}_d$ is the desired length of the distance constraint, $\mathbf{L}_0$ is the original length of this constraint, and $\mathbf{a}$ is the interpolated activation for the muscle in the current simulation step.

$$\mathbf{L}_d = \mathbf{L}_0.max\{1.0 - \mathbf{a}, 0.7\} \tag{5.11}$$

# Model implementation

# 6

The proposed model was implemented in the context of the **Muscle Wrapping 2.0** project, which is an OpenSim plugin to simulate a musculoskeletal system. During the implementation, attention was paid to the flexibility, readability and efficiency of implementation. In comparison with the PBD implementation [KČ21], greater effort has been made to utilise the capabilities of modern *C++* programming language and well-optimised *C++* libraries. The aim was also to create a well-structured basis for future research on this method, allowing for extensions in the forms of novel concepts to be easily introduced to the system.

The implementation is **partially based** on the former approach of the PBD method [KČ21]. Mainly in the part of parsing outer structures to internal structures.

The description of the implementation will follow a chronological order of the development and explanations of decisions made along the way regarding unexpected hurdles and attempts to solve them.

## 6.1 Attempts to keep muscle-bone proximity

At the very beginning of this project, in the context of a precursor course to this thesis, the **KIV/OP** in the year 2022 at the University of West Bohemia, Faculty of Applied Sciences, an attempt to keep the muscles near the bones was made using an SDF, effectively letting the muscles slide across the surfaces of the bones. The structure keeping track of these trapped muscle vertices is shown in Source code 6.1. This structure is a map containing for each bone a vector of vertices which are trapped in the proximity of the bone. Together with the muscle vertex index, the original distance to the bone surface is also kept. This structure is built at the beginning of the simulation. During the simulation, each of the trapped muscle vertices is inspected to determine whether or not it has been distanced too far from the bone surface. If so, it is pushed towards the bone so that the original distance is kept, while being able to deform in other directions, resulting in a sliding motion.

Source code 6.1: Structure

```
std::map<size_t, std::vector<std::pair<size_t, double>>>
    m_SDF_trapped_points{};
```

As an alternative, the idea of introducing virtual PBD distance constraints between the muscles and the bones came. Currently, this concept was re-purposed for solving collisions between elastic muscles.

On top of that, to obtain a better understanding of the PBD solver, visualisation of the vertex displacements by constraint per inner iteration for one outer iteration was made, keeping the position advancements in a quite complex structure, which currently looks like Source code 6.2 shows. This structure keeps a collection of forces for one outer solver iteration. The forces are represented by a vector (element for each sub-iteration) where the element is a snapshot of all the vertex positions and the corresponding **constraint_type** of force (produced by a constraint) that caused this displacement. These constraint forces are visualised in different colours so that visual distinction between the constraint type influence in a particular area can be made.

Source code 6.2: Structure of the constraint displacement forces to visualise

```
typedef
std::vector<std::pair<std::vector<Eigen::Vector3d>,
    constraint_type>> forces;
```

The results of the **KIV/OP** project were partially showcased and discussed by a conference paper [Čer+23] in the year 2022.

## 6.2 First XPBD, solver acceleration and virtual edges

Later, the coursework of the subject **KIV/PPR** concerning parallel programming was joined with the topic of this thesis. An attempt was made to extend the PBD algorithm [KČ21] into the XPBD algorithm, while keeping the original structure of it. But soon, this approach showed significant drawbacks, as the resulting implementation was too complex and hard to understand.

Many acceleration strategies were implemented, like refactoring the structures for **better sequential spatial data locality** with **direct access**, while also allowing better **memory contiguity**, since the arrays of structures were refactored to a structure of arrays, where **SIMD vectorised operations** can be utilised. Tactics like **unrolling of the loops** were used to accelerate the bottlenecks of constraint projection, and the parallel C++ standard library was employed in various places for CPU parallelisation.

The first **proof-of-concept** of using **virtual edges for muscle collisions** was implemented in many modalities (on the CPU, on the GPU, partly on the GPU, and minimally on the GPU), all of which were implemented to run in parallel. The CPU parts of the implementations were also able to switch to run sequentially, to better compare the results of the efficiency and functionality of each approach.

The proof-of-concept showed, that unless the whole solver runs on the GPU, unnecessary copying mainly of the positions is not amortised by any of the GPU implementations. On top of that, the sequential, progressive avoidance of found contact distance proximities typically showed the best results in terms of collision handling. The algorithm of the collision handling will be disclosed later in this text in its current state.

From this point onwards, all the corresponding described methodologies (Chapter 5 on page 51) and implementations were designed and completed in the course of this thesis.

## 6.3 Extended Position Based Dynamics

The task was to design the XPBD deformation algorithm (described in detail in Section 5.1 on page 52) all over from the start while separating it from the PBD method. While the basic method algorithm 3.1 stays very similar (with the change of the inner loop 3.2), the constraints were chosen to be represented by the **Curiously Recurring Template Pattern** (cppreference.com/w/cpp/language/crtp), and the **Eigen** C++ template library for linear algebra (eigen.tuxfamily.org) was utilised.

The implementation of the XPBD algorithm and constraints was done according to the proposed designs shown by Figure 5.1 on page 53 and Source code 5.1 on page 55, respectively. In addition, during the **init()** method, the muscle handling instance and the graph instance are not the only things created. Furthermore, in this method, the SDFs using Discregrid are also created for each bone, some of the muscle vertices are fixed to an appropriate bone, and all the constraints are generated.

### 6.3.1 Constraints

The distance and dihedral constraints were implemented as *soft* constraints, meaning non-zero compliance and the presence of the constraint dissipation potential regularisation. The volume constraints have been implemented as *semi-hard* constraints since their compliance is very close to zero (e.g. $1e^{-10}$), but are not regulated by the vertex velocities. The *hard* constraints (infinite stiffness, no compliance) are the collision constraints.

For the implementation of **Rayleigh's dissipation potential** constraint regularisation for the distance and dihedral constraints (in the CRTP), the current velocities

of the concerned particles are needed. These velocities need to be updated just once before the first inner solver iterations, which is done explicitly when re-setting the Lagrange multipliers, without keeping the references to the vertices in each constraint.

In the CRTP, the parent class called **constraint** provides a method **void project()**, the full body of which is provided in Source code 6.3. Most of the variables are kept as class members to prevent unnecessary memory allocation. In the source code, line **(1)** casts its reference to the reference of the derived instance, which on line **(2)** provides the current cost of its objective function. Sometimes, the cost does not make sense to compute, e.g. when it exceeds physical limits or zero division occurs. In that case, the deriving instance should return the constant **ms_do_not_solve** to avoid undefined constraint behaviour or redundant computation of the gradient (perhaps the cost is nearly zero, meaning the constraint is already satisfied). On line **(8)**, the deriving class instance provides the gradient of this constraint. Lines **(9-12)** compute the change of Lagrange multiplier using time-weighted compliance **m_alpha_tilde**, previous **m_lambda**, the regularisation term **m_gamma** and its pre-computed plus one version, along with the vector of current simulation step velocities. These lines are the implementation of Equation 3.7 on page 37. In the next line, the Lagrange multiplier is accumulated to the current outer iteration sum. The line **(14)** computes Equation 3.6. Lines **(16-17)** project the displacements onto the particle references.

Source code 6.3: Implementation of the constraint projection method

```
1  void project()
2  {
3      const auto derived = (static_cast<const Derived*>(this));
4      const double cost = derived->cost();
5      if (cost == ms_do_not_solve)
6          return;
7
8      derived->grad(m_grad.data());
9      const auto grad_transposed = m_grad.transpose();
10
11     const double delta_lambda = (-cost - m_alpha_tilde *
       m_lambda - m_gamma * grad_transposed *
       m_flattened_velocities) / (m_one_plus_gamma *
       grad_transposed * m_grad + m_alpha_tilde);
12
13     m_lambda += delta_lambda;
14     m_delta_x = m_grad * delta_lambda;
15
16     for (size_t i = 0; i < ParticlesNumber; ++i)
17         m_particles[i].get() += m_delta_x.segment(3 * i, 3);
18 }
```

The proposition of **parallelization** in section 5.1.3 on page 55 was also implemented. In particular, during the creation of the constraints a map is also created, where the key is the index of the influenced vertex, and the value is a container of all constraint indices of that particular type that influence this vertex. Then, out of this *vertex_to_constraint* mapping, the constraint graph is created, where the constraints are the nodes and an edge exists between the constraints if they share an influenced vertex, which is the information in the map. This graph is then coloured greedily using the SLO pre-sorting of vertices based on their degrees, and finally, for each found colour, the corresponding constraints of that colour are assigned to a colour group, which can later be run in parallel using PSTL. Volume constraints are trivially parallelizable and were not further accelerated, as they did not show to be the efficiency bottleneck.

On the other hand, efficiency tests done along the implementation using the **Visual Studio Profiler** kept showing that the true bottleneck is the dihedral angle constraint projection. With the use of Eigen, the most problematic part was the calculation of ***arcus cosinus***, which has been solved by **approximating** this function with the function provided by Nvidia at the web page developer.download.nvidia.com/-cg/acos.html with an approximate error of just $\approx$ **0.00384**°. During the implementation of the dihedral constraints, another problem occurred, resulting in the **muscle explosion**.

### 6.3.2 Major structures

The main properties the algorithm needs are kept in containers of **Eigen::Vector3d** instances, as the space of the musculoskeletal system is expected to be three-dimensional. These are the currently predicted positions **p**, velocities of these particles with the same indexing **v**, currently acting forces **f**. The constraints may also generate some additional forces (e.g. the collision constraints could generate a proportional force on the penetrating vertices in the opposite direction of their movement) the solver would propagate into the next outer simulation iteration called **previous_f**. This additional force propagation was implemented as a reaction to an emerging problem of **rapid muscle motion**, which will be described in detail later in the text. These properties belong directly to the XPBD simulator instance, hence no indirect access is done when traversing them in the solver methods. Note also the **structure of arrays** scheme of the members, as opposed to representing the PBD particles explicitly and traversing them with inefficient stride due to inefficient memory contiguity, etc.

Sadly, since C++ provides no efficient container that would be able to hold differently templated parent class instances, the constraints have to be kept in separate

containers, as shown in Source code 6.4. It is important to remind the reader, that the volume constraints (per each muscle mesh of various influenced vertex numbers) are not subject to the CRTP scheme due to the drawback, where the CRTP deriving constraint class must specify a fixed number of particles it influences, which, in case of the volume constraint, is mostly different for each of the constraints.

Source code 6.4: Containers of the constraints

```
1 std::vector<distanceConstraint> m_distance_constraints;
2 std::vector<dihedralAngleConstraint>
    m_dihedral_angle_constraints;
3 std::vector<volumeConstraint> m_volume_constraints;
```

### 6.3.3 More muscles

As this implementation allowed for more inner solver iterations (for example 70) while keeping the constraint projection consistent, the musculoskeletal model has been enriched by 16 new possible muscle geometries, together with their insertions and origins, to be able to observe the XPBD deformation behaviour in more situations. This data was sourced from the **LHPBuilder VPHOP WP10 DEMO** application downloadable at the address mi.kiv.zcu.cz/en/research/musculoskeletal.html.

The added muscles include the *adductor longus*, the *adductor magnus*, the *gluteus minimus*, the *gracilis*, the *obturator externus* and *internus*, the *pectineus*, the *piriformis*, the *psoas*, the *rectus femoris*, the *sartorius*, the *semimembranosus*, the *semitendinosus*, the *tensor fasciae latae*, the *vastus lateralis* and *medialis*. The muscles together with the already present *iliacus*, *glutei maximus* and *medius*, and *adductor brevis* are all rendered in Figures A.1 and A.2 from different views. All these muscles belong to the right side of the body and should cover all the muscle groups used for various movements of the lower extremity in the hip area.

For some of the muscles, some attachment areas were not available in the downloadable dataset. To be able to use these muscles in the system during deformation, the missing attachment areas were picked by hand in the **Blender** (www.blender.org) 3D modelling tool.

As the plugin defines an **XML** setup file, where the muscle geometries and attachment areas are defined, each of these hand-picked, **anatomically inaccurate** attachment areas, has a warning comment associated, hopefully alerting the user from making any physiological inferences using these attachments. A configuration named **setup_MuscleGeneratorTool_All_Hip.xml** has been created to incorporate all the muscles.

With this addition to the scene, in a similar spirit as the unnatural bending of the *iliacus*, some of the **muscles attached to the distal part of the femur or even below the knee**, attached e.g. to the *tibia*, **would lag behind the movement of**

**the bone** greatly, in a wave-like fashion. This situation will be discussed in the next chapter.

## 6.4 Muscle collision handling

Since the proof-of-concept showed, that an iterative refinement of the positions might be beneficial to early collision avoidance, muscle collision handling was implemented using the virtual edges processed sequentially, while in case the CPD is met by the current virtual edge, it resolves it right away, changing the geometry of both muscle surfaces, allowing the next processed virtual edge to help avoid collisions in different areas, or refine the position the previous virtual edge tried to solve.

Two approaches to avoiding the collisions were implemented, where one works by pushing each vertex of the virtual edge of length less than CPD away from each other by the shortest distance possible so that they stay in that CPD distance. The other approach is more complex as it also uses information about the particular virtual edge starting position. The algorithm is described in the section 5.2.1 on page 58. Both the simple resolution (avoidance) and the more complex one are described in the section 5.2.2 on page 64.

Since unexpected pitfalls with the simple collision resolution were identified (example given in Figure 5.10), the default implemented approach is the one **concerning the previous positions of the virtual edge**, according to Equations 5.3 and 5.4.

The algorithm of the Zig-Zag iterations of virtual edges can be seen in Source code 6.5. Definition for virtual edge $i$ is taken from the two containers on lines **(3-4)**, where one container at a given index contains the virtual edge $a$ point in one muscle mesh, and the other container, at the same given index, contains the node $b$ for the same virtual edge, belonging to another muscle mesh. These indices of muscle nodes are saved on the lines **(5-6)** as the former ones, later used for resolution. Next, on the lines **(7-11)**, the current squared distance of the positions on the indices $a$ and $b$ is computed, and loop-controlling variables are prepared. At line **(12-13)**, the Zig-Zag loop does not stop until a local optimum is reached or the edge has "collided" (met CPD), then the iterations can also end by reaching the user-defined maximum iterations. Line **(15)** represents the *Zig* step, while line **16** represents the *Zag* step while continuing with the possibly changed distance returned by the first step, *L2_a*. A local minimum is reached **(18)** when the length of the edge has not changed with *Zig* nor *Zag*. Comparing the **double** values like this can fail due to numerical errors and floating point number representation but is left like this for the sake of simplicity. On line **(20)**, the distance is updated to the last one computed, and on the next line, a contact proximity distance test is made. At the end of the Zig-

Zag iterations, the possibly newly found indices for this virtual edge are updated to the global containers on lines **(22-23)**, and if a collision occurred (the virtual edge reached the minimum allowed CPD), this collision is resolved concerning the former virtual edge position.

Source code 6.5: The Zig-Zag algorithm

```
1  for (size_t i = 0; i < m_virtual_edge_count; i++)
2  {
3      uint32_t a = m_virtual_edges_a[i];
4      uint32_t b = m_virtual_edges_b[i];
5      const size_t a_former = a;
6      const size_t b_former = b;
7      const auto diff = m_ps[a].get() - m_ps[b].get();
8      double L2 = diff.squaredNorm();
9      bool collided = false;
10     bool local_optimum = false;
11     uint32_t iter = 0;
12     while (!(local_optimum || collided) &&
13                              iter++ < m_max_iter)
14     {
15         double L2_a = find_minimizing_neighbour(&a, b, L2);
16         double L2_b = find_minimizing_neighbour(&b, a, L2_a);
17
18         local_optimum = (L2_a == L2) && (L2 == L2_b);
19         L2 = L2_b;
20         collided = L2 < CPD_SQUARED;
21     }
22     m_virtual_edges_a[i] = a;
23     m_virtual_edges_b[i] = b;
24
25     if (collided)
26         resolve_collision_with_respect_to_prior(a, b,
27                                  a_former, b_former, L2);
28
29 }
```

The **find_minimizing_neighbour(&a, b, L2)** method checks the topological neighbours of node **a**, moving the node **a** to their position would make the squared distance **L2** lesser. The **L2** is the squared distance to the referential point with index **b**. The **resolve_collision_with_respect_to_prior(a, b, a_former, b_former, L2)** method implements the collision resolution according to Equations 5.3 and 5.4.

## 6.5 Angle-driven muscle interaction

The last implemented part of this thesis is the realisation of muscle interaction through synchronous contractions based on bone orientations. First, the activations

must be acquired. For that purpose, OpenSim of version **4.5** Static Optimization Tool was used.

### 6.5.1 Activations acquisition

To run the SOT, a model must be specified. A model that has been used for activations acquisitions in this thesis is the model done by [Raj+16] and later updated by [Uhl+22], as this model contains the majority of the muscles needed for various motions influencing the hip area. This model also comes with the basic OpenSim 4.5 package download.

Next, the model needs a motion to execute. The example picked for this task is the hip flexion. To describe the motions of the bones, OpenSim accepts the format **.mot**, which begins with a header in the following format:

```
hip_flexion     # The name of the motion
version=1
nRows=11        # Number of table rows
nColumns=10     # Number of table columns
inDegrees=yes   # Angles defined in...

Units are S.I. units (second, meters, Newtons, ...)
Angles are in degrees.

endheader
```

Table 6.1: Example motion file table for flexion of the right hip

| time | ... | hip_flexion_r | hip_adduction_r | hip_rotation_r |
|---|---|---|---|---|
| 0.05 | ... | 0 | 0 | 0 |
| 0.055 | ... | 1 | 0 | 0 |
| 0.06 | ... | 2 | 0 | 0 |
| 0.065 | ... | 3 | 0 | 0 |
| 0.07 | ... | 4 | 0 | 0 |
| ⋮ | | ⋮ | ⋮ | ⋮ |
| 0.50 | ... | 90 | 0 | 0 |

After that, a table begins. The columns describe different types of motions of the bones and even transformations of the whole system, and each row represents a time entry with values of each bone movement. For example, a table for pure right hip flexion from 0.05 to 0.5 seconds would contain non-zero values only for the columns of **time** and **hip_flexion_r** which would look like in Table 6.1. The **time** column is always the first one and its values should form a non-decreasing sequence.

This table can be loaded into the OpenSim **Storage** object (simtk.org/api_docs/-opensim/api_docs/classOpenSim_1_1Storage.html), where each row of the table corresponds to one so-called **state vector** of the **Storage**.

The last thing that is needed is the file defining settings for actuators. In all of the measurements of activations presented for this thesis, the actuators from controlled muscle control (CMC) for walking dynamics, supplied with this model, were used.

The tool, given such actuators, can be run with the use of the muscle force-length-velocity constraint function. When the SOT tool is done, the muscles on the model should be coloured from blue to red, describing how much the muscle is activated in the current pose. Blue muscles are passive, while the more red the muscle, the more it activates. The animation of the movement can be played to see the dynamics of the activations during the movement.

An output file is generated with the suffix **StaticOptimization_activation.sto**. This file has the same structure as the **.mot** file described above, while the table 6.1, instead of having the motions of joints as columns, the columns are now activations for a particular muscle. One row represents a vector of activations for all model muscles for a specific time (the same one as in the input motion file).

The file structure that has been chosen in this thesis for joining the movement information with the activations is the following. The first column is the **time**. The next three columns describe the motion of the bone (e.g. **x**, **y**, **z**), what follows are the columns for the model muscles, containing activations corresponding to the given bone movement. It is the operation of joining the two tables by the **time** column.

What might be problematic, is that the model often contains e.g. three different representations (fibres), and therefore on the output, there are now three columns for one muscle, while in the **Muscle Wrapping 2.0** musculoskeletal model, this muscle is represented as a mesh. One option could be to partition each muscle by these fibres, but that is beyond the scope of this thesis. A simple **sum** across these three columns is used to generate one activation column for each split muscle. This could be feasible even from the bio-mechanical point of view, since in the first place, the activations were distributed across these three fibres by the optimization process.

A *Python* script to look through two sub-folders, where if one contains the **.mot** motion files, the other contains the resulting **.sto** files, where the corresponding files have the same name, the script joins all these into one big table, where the **time** no longer matters (as the activations correspond to the bone motion columns). The split activations are summed together, and the motions themselves are joined by rows. Given the muscle names of the columns correspond to an expected nomenclature, this file now represents the samples discussed in Section 5.3.2.

### 6.5.2 Activations as the input of the system

As previously noted, the **Muscle Wrapping 2.0** is a plugin to the **OpenSim** software, which defines an **XML** file, where apart from the muscle geometries, motion file definition, etc. One of these definitions is for the **kinematics_fibre_algorithm**. A new possible entry has been created for the XPBD deformation method in this thesis. An example of the entry is given in Source code 6.6.

Source code 6.6: Kinematics fibre algorithm specification for muscle interactions around hip joint

```
<kinematics_fibre_algorithm>
  <Havlicek2024XPBDAlgorithm>
    <joint_interaction_information>
      <!-- Information describing the interactions of a joint
     under various angles. -->
      <JointInteractionInformation name="hip_r">
        <body>femur_r</body>
        <!-- The position of the physiological joint centre.
    -->
        <centre>143.797 243.457 -432.585</centre>
        <!-- The orientation of the physiological joint
    centre. -->
        <orientation>-0.3626 -1.5127 -2.0524</orientation>
        <!-- Scale factors of the centre coordinates in X, Y,
     and Z directions respectively. -->
        <scale_factors>0.001 0.001 0.001</scale_factors>
        <!-- Muscles involved in the movements of this joint.
     -->
        <muscles_involved>
          GluteusMaximus GluteusMedius GluteusMinimus
    Piriformis Sartorius TensorFasciaLatae Gracialis Iliacus
    Pectineus Psoas Semitendinosus Semimembranosus
    BicepsFemoris RectusFemoris AdductorMagnus AdductorLongus
    AdductorBrevis
        </muscles_involved>
        <!-- A .sto file containing the activations of
    muscles for various angles. -->
        <muscle_activations_file>hip_activations.sto</
    muscle_activations_file>
      </JointInteractionInformation>
    </joint_interaction_information>
  </Havlicek2024XPBDAlgorithm>
</kinematics_fibre_algorithm>
```

In the example source code, the **Havlicek2024XPBDAlgorithm** element presence specifies that the deformation system presented in this thesis will be used. This algorithm element has only one child, which is a list of **joint_interaction_information**.

This list can contain as many joint interaction (JCU) definitions as desired, but as of now, only one such entry is defined with the element **JointInteractionInformation**, containing the name of the joint suffixed by the indication of the body side this joint belongs to, in this example **hip_r** specifies the right hip joint. The **JointInteractionInformation** element contains six child elements:

- **body** - the transformation object in the scene to which the joint should be attached
- **centre** - the physiological centre of the bone inside of the joint, currently used only for visualisation purposes
- **orientation** - the physiological orientation of a bone inside of the joint (in this case this is the orientation of *femur* head), currently not used
- **scale_factors** - the scale the **centre** should be subjected to
- **muscles_involved** - the list of names of muscles (these names are given to the muscle definitions further in the **XML**) that should be involved in the movement
- **muscle_activations_file** - the path to the samples **.sto** file generated in the previous section

The **centre** and the **orientation** in this example were obtained using the **msk-STAPLE** Matlab tool (github.com/modenaxe/msk-STAPLE) done by Modenese & Renault in 2021 under the **CC BY-NC 4.0** license (creativecommons.org/licenses/-by-nc/4.0).

This algorithm entry is then appropriately parsed by the system to a corresponding OpenSim plugin model definition, also created in the context of this thesis. The model is then passed to the deformation algorithm itself, and interaction joint representations are created.

### 6.5.3 Internal JCU representation

On the initialisation, the XPBD deformation algorithm receives a container of parsed models representing the information needed to facilitate joint-governed muscle interactions. On top of the information that is found in the previous section, the XPBD method also receives for each JCU, the whole information about the motions of this JCU's bones during simulation (extracted from the **.mot** file), so that the JCU knows which orientation to interpolate for at a given simulation step.

From this container, an internal container is created, with the instances of the **XPBDJoint** class, responsible for the muscle contractions and relaxation controls

(a relaxation of a muscle in this context means that a muscle is leaving a contraction, as the desired lengths of contraction distance constraints become longer, up to their original lengths) around a particular JCU during its movement.

The JCU keeps track of a plethora of information, as has been described in Section 5.3.1 on page 67. The major members are to be seen in Source code 6.7.

Source code 6.7: Major member variables of the **XPBDJoint** class

```
1 vtkSmartPointer<vtkKdTree> m_tree;
2 std::vector<std::vector<distanceConstraint>>
    m_contractions_by_muscles;
3 const std::vector<std::vector<double>> m_activations;
4 const std::vector<Eigen::Vector3d> m_angles;
5 const std::vector<Eigen::Vector3d> m_motion_data;
6 const std::vector<std::string> m_muscle_names;
7 std::vector<double> m_current_activations;
8 std::vector<double> m_current_weights;
9 const std::string m_body;
10 static constexpr size_t s_K_neighbours = 10;
```

Starting from the **vtkKdTree** pointer on line **(1)**, this is a reference to an implementation of the kd-tree provided by the **Visualisation Toolkit** (VTK, vtk.org). VTK is the toolkit **Muscle Wrapping 2.0** uses to manipulate and visualise the 3D data, e.g. all screenshots from the simulation in the following chapter are done using this toolkit and the **vtkVisualDebugger** library (gitlab.com/besoft/vtkVisualDebugger) done by Kohout & Hájková in 2009 under the **APACHE LICENSE, VERSION 2.0** (apache.org/licenses/LICENSE-2.0). The kd-tree is created in the **XPBDJoint** constructor out of the **m_angles** vector on line **(4)**, representing the sequence of the **sampled** JCU bone orientations. The kd-tree uses these angles as a point cloud, as visualised in Figure 5.18 on page 76. On line **(3)**, the member represents the table of **sampled** muscle activations corresponding to the angles. On line **(5)**, the member stands for the sequence of the orientations of this joint for each simulation step. Line **(6)** contains the names of the muscles involved from the **XML** element. Line **(2)** represents the vector of involved muscles, where each muscle has a vector of distance constraints associated with it, representing the detailed surface fibres. These constraints are generated for this JCU from the detailed surface fibres if an involved muscle the surface fibre belongs to is one of the muscles involved in this JCU. Line **(9)** contains a **string** representing the transformation object this JCU is attached to, used for deciding if this JCU should update the desired lengths of muscle contraction distance constraints during the simulation. Line **(10)** defines the number of closest neighbours which should be interpolated for. Lines **(7-8)** represent the activations and weights, respectively, of all the **sampled** muscle activations, used primarily for visualisation purposes.

The interpolation of involved muscle activations and subsequent modification of desired lengths of contractile distance constraints is done once per outer solver iterations, as the JCU bone orientations change. The joint can project its constraints by calling the **project()** method 6.3 on each of them. The JCU contractions are projected as any other constraint, in the loop defined in Source code 3.2 on page 36.

## 6.6 Problems encountered along the way

During the development, various problems arose. Some of the problems may have stemmed from the algorithms proposed, others due to lack of detailed information in a particular area.

### 6.6.1 Detailed surface fibres running perpendicular

The observation made in Section 5.3.1.1 of the surface fibres being generated in the wrong directions can be visualised directly on the muscle data, as Figure 6.1 shows for the resulting detailed surface fibres of the *gluteus maximus* (in dark greed for better contrast). The figure shows the majority of fibres in the muscle's centre being generated well, but the ones that are near the lateral part of the muscle (on its right *edge* from the dorsal view), a lot of the fibres must cross to the other side of the muscle, where the next nearest point to the inner fibre was found.

This may not be so problematic if there are also correct fibres running along the area, but consider the following Figure 6.2. In the figure, a blue *iliacus* muscle is

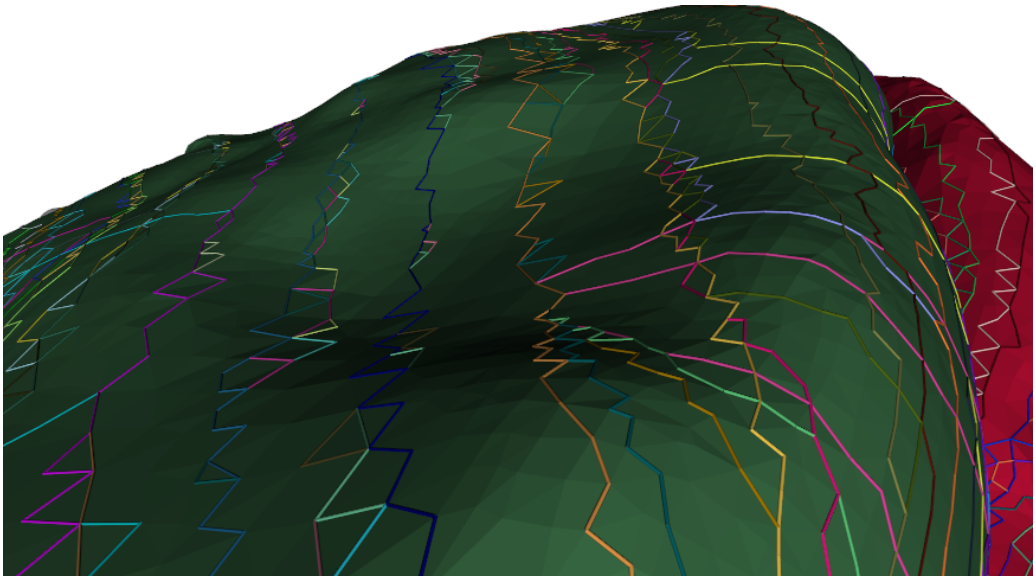

Figure 6.1: Perpendicularly running detailed surface fibres

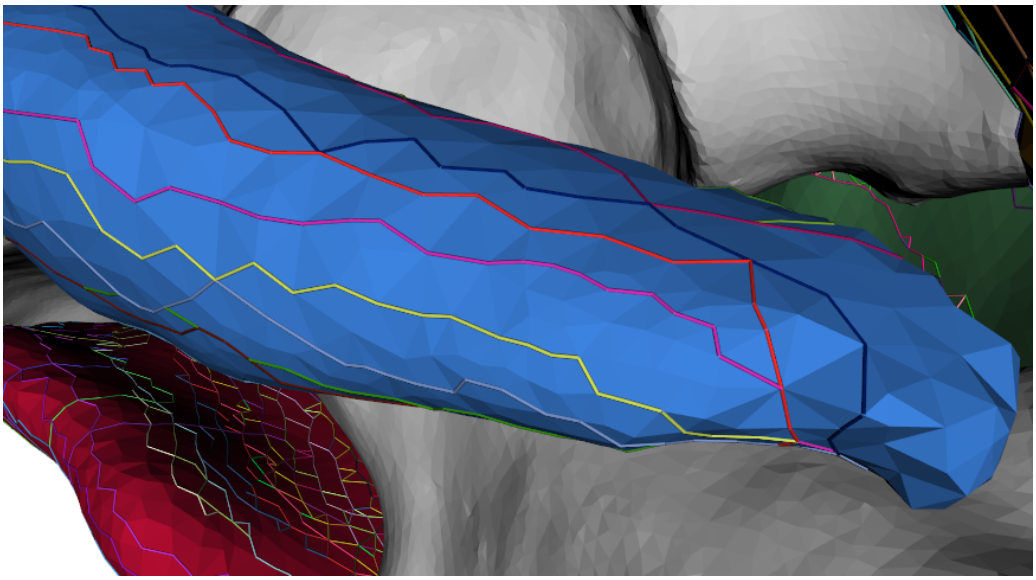

Figure 6.2: *Iliacus* insertion area not covered by fibres using low-resolution inner fibres

shown, covered in just a few detailed surface fibres. The number of fibres is a parameter to the system (when defining the muscle in the **XML**), and should be **carefully considered when using this method of detailed surface fibre extraction** since big areas of the muscle could be left out. The more problematic area in the figure is the insertion into *femur*. There, the same situation as illustrated in Figure **??** on page **??** happens, where the fibres also skip a bulk of the muscle due to the majority of fibres, which should cover it, running in perpendicular to the inner muscle fibres, instead. This problem can be solved by **making the resolution of the fibres higher**, which is the second parameter in the configuration. Otherwise, the muscle volume will unrealistically accumulate in this area during a contraction.

A possible solution to this problem is not only the usage of higher inner fibre resolution and count, but the fibres can also be **picked by hand**, which should generally result in more plausible contraction results. The last possible solution to this problem could be to use a different method for obtaining the surface fibre details, e.g. with the **shortest geodesic distance path search**, which has been previously implemented in the **Muscle Wrapping 2.0** project.

### 6.6.2 Muscle explosion

The explosion of deformed muscles can happen for multiple reasons. In this section, one of the cases will be described, specifically the case of *gluteus maximus* under a high degree of flexion (**90**°).

Figure 6.3: Progression of edge penetration near muscle attachment area (from left to right)

#### 6.6.2.1 Problem description & Possible causes

Looking at the first detail of its insertion into *femur*, at approximately **80**° of flexion (Figure 6.3 on the left), a few visible triangle edges penetrate the bone near the attachment, as the attachment is pulling onto the muscle, while it tries to preserve its shape. This is an expected behaviour, at least to an extent. However, the resulting degenerated triangles, as will be uncovered, can result in a bigger problem. For clarity, the view is situated from the attachment towards the *femoral* neck with lowered bone opacity.

What can be seen at the same time from the inside of the muscle (centre image in Figure 6.3), looking in the opposing direction, it is obvious, that some vertices were left on the other side of the bone, while most of the rest of the surface is embracing the bone well, resulting in the edges-bone penetration. It should be noted, that no visible vertex-to-bone penetration occurs.

Looking at the view from above of the attachment (right image in Figure 6.3), it can be seen that the vertices on the other side are the ones attached to the *femur*

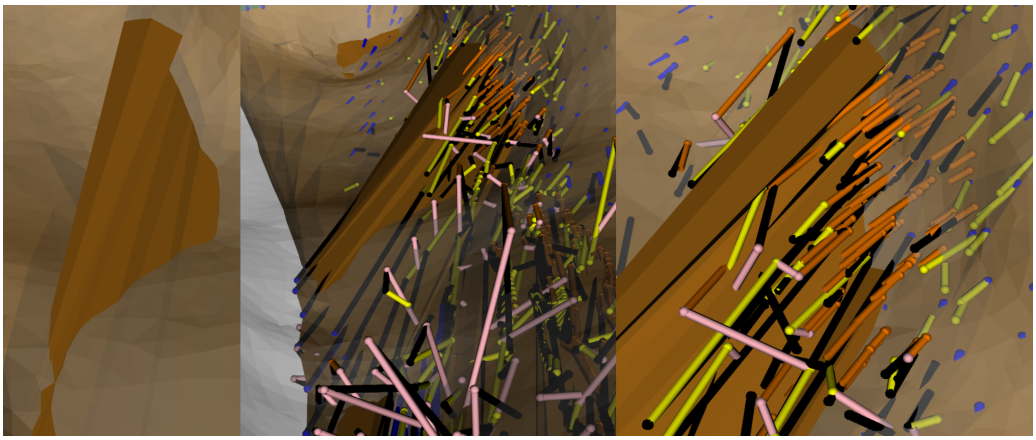

Figure 6.4: Edge penetration and internal constraint forces at work (from left to right)

(highlighted in blue polyline).

As the flexion continues (left image in Figure 6.5) up to approximately **87**°, these degenerated triangles get longer, as the attached vertices keep being pushed, while the vertices on the other side keep being kept from the bone, being constrained mainly by the collision constraints, as well as the distance and the dihedral angle ones. The number of penetrating edges grows with the flexion, as even the attached vertices further down the insertion get estranged from their original surface area, which is constrained on the other side of the bone.

Taking a closer look at the constraints acting on the interface between the bone and the muscle surfaces (centre and right image in Figure 6.5), it can be seen, that the yellow distance constraints are trying to correct the vertices towards the attachment, and the black contraction distance constraints, trying to achieve the same. Acting in the opposite direction, the orange displacements representing the bone collision response are keeping them on the bone surface. At the same time, the dihedral angle constraints (in pink) are desperately trying to fix the nearly planar deformation, counter-productively pushing the whole muscle neighbourhood inside of the bone.

As this continues, the dihedral constraints become unstable, as can be seen in the top image in Figure 6.4, pushing the vertices inside of the bone. The instability

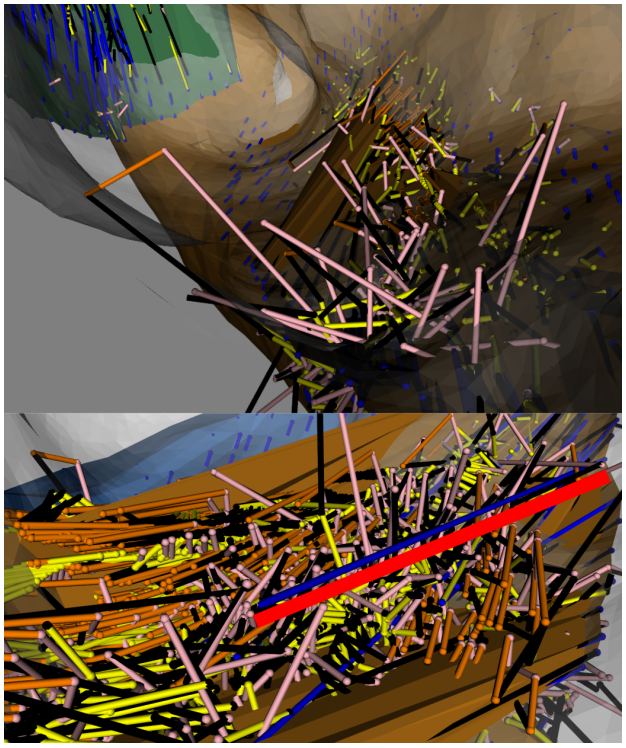

Figure 6.5: Inspection of the forces (top and bottom) leading up to explosion (on the bottom) under flexion bigger than **90**°

eventually causes at least one of the vertices to finally satisfy some of the dihedral angle constraints, the position of which can get instantly pushed to the other side of the bone, generating a great internal force (in blue, highlighted by the parallel red line in the lower image of Figure 6.4), leading up to the explosion of the muscle, eventually the tunnelling of it to the other side.

This is not the only situation where the problem can emerge. Consider, for example, the attachment area which is, instead, pushed into by a bone. Easily, the dihedral angle constraints could all of a sudden gain great cost, as the angles between triangles may even invert this way. Perhaps in any situation, where the dihedral angle constraints are largely violated and outweigh the distance constraints (which try to prevent the explosion), the dihedral constraints may start the explosion of the muscle, trying to resolve their original angles.

### 6.6.3 Possible solutions

Trying to fine-tune the system to prevent this behaviour globally would most probably go at the expense of deformation quality in other places. Nor is the solution to turn off the tunnelling detection, since the problem gets only postponed (as the muscle mesh is then allowed to slide across the bone surface a little). This problem is the result of all the constraints acting in response to each other. Compared with the previous PBD approach [KČ21], this situation probably emerges because of the pronunciation of the XPBD constraints, making the muscles react more lively to the bone movements, which could otherwise be a desired property of the deformation. Nevertheless, it may be the cause of this emerging problem.

One of the plausible solutions could be to prevent the edges from penetrating the bone in the first place. As one of the edge vertices would be fixed, the other one would be forced to resolve the collision. In the example shown above, the *gluteus maximus* could potentially be prevented from exploding, since the muscle surface could be forced to move further underneath the *femur*, preventing the formation of degenerated triangles and the following crude violation of the dihedral angle constraints.

Another approach could be to prevent the dihedral angle constraints from acting on degenerated triangles or keep them from trying to fix extremely deformed angles, the latter of which has been implemented in this thesis simply by not solving the dihedral angle constraint if its cost is unreasonably large. However, this does not solve the problem completely.

Lastly, the number and stiffness of the contraction distance constraints can be reduced, since these constraints not only facilitate the contractions and relaxations but also duplicate the basic distance constraints behaviour, if no contraction happens. The solution could be to not project those contraction distance constraints, which

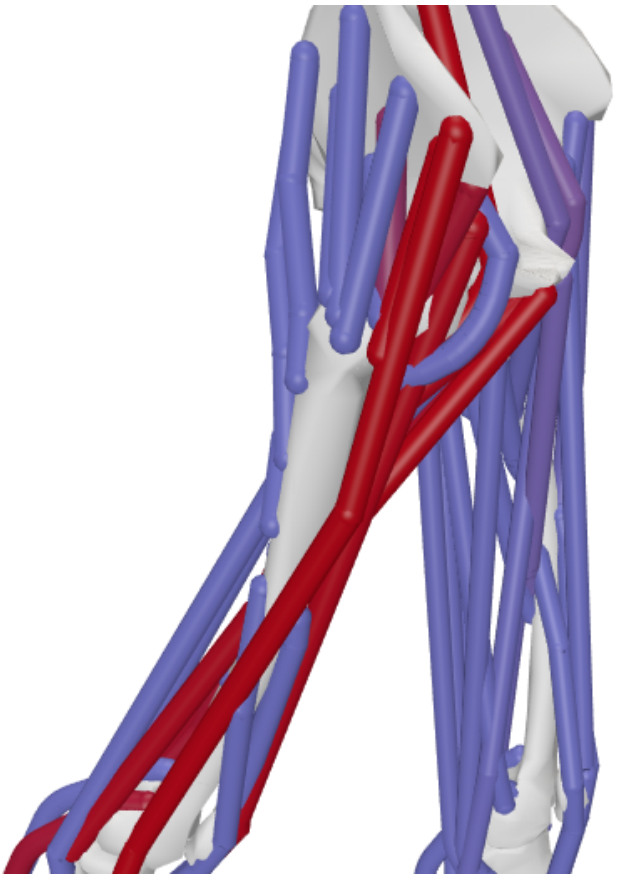

Figure 6.6: OpenSim SOT results, where muscles counter-intuitively contract during 40° hip extension

are desired to preserve their original lengths, but only those, where contraction takes place.

### 6.6.4 Pelvic tilting during activations acquisition

On the output, the OpenSim SOT seems to give, at first glance, counter-intuitive results near the limits of ranges of motion. At the limits, suddenly, muscles that should be the most relaxed, start to contract. But as has been discussed in Section 2.3, **a lot of the muscles play an important role in stabilising the *pelvis* and the joint**. This is the proposed explanation for this kind of behaviour. As will be described in the following chapter, these activations will cause undesired muscle deformations. A screenshot from the application is given in Figure 6.6, where the most activated muscles are in red, and the relaxing ones are in blue. The red muscles are probably trying to stabilise or actuate the *pelvis* tilt caused by the extreme hip extension.

### 6.6.5 Stiffness tuning

To tune the main parameters of the XPBD deformation method (which are the constraint stiffnesses) for the specific purpose of simulation muscle behaviour was a challenging task. To explain the process, Equations 6.1-6.3 remind the reader of the equations of constraint projection using the XPBD method, where $\alpha$ is equal to one over the constraints ($\frac{1}{k}$) stiffness and the time step of the simulation $\Delta t = 0.001$.

$$\tilde{\alpha}_j = \frac{\alpha}{\Delta t^2} \tag{6.1}$$

$$\Delta\lambda_j = \frac{-C_j(\mathbf{p}_i) - \tilde{\alpha}_j\lambda_{ij}}{|\nabla C_j|^2 + \tilde{\alpha}_j} \tag{6.2}$$

$$\Delta\mathbf{p}_i = \nabla C_j(\mathbf{p}_i)^T \Delta\lambda_j \tag{6.3}$$

If, for example, the distance constraint $j$ between two vertices $a$ and $b$ is considered, then the cost of the constraint is defined as the next equation says.

$$C_j(\mathbf{a}) = C_j(\mathbf{b}) = \|(\overrightarrow{\mathbf{ab}})\|_2 - \mathbf{d},$$

where **d** is the desired length (in this case, the original length). Now consider that the positions of vertices in the **Muscle Wrapping 2.0** project are often expressed in meters. The computed average distance between two neighbouring vertices is therefore roughly equal to 0.00235 meters (computed across 11 different muscle meshes). If this is the original length of the edge (**a**, **b**), and it is prolonged two times, therefore its current length is equal to 0.0047 meters, then the cost for each vertex is equal to the following expression.

$$\|(\overrightarrow{\mathbf{ab}})\|_2 - \mathbf{d} = 0.0047 - 0.00235 = \mathbf{0.00235}$$

With this mathematical basis, it can be inferred and experimentally observed, that during the simulation, unless the constraint stiffness is set to a very large number, e.g. $\mathbf{1e^{15}}$, then the change in the Lagrange multiplier $\Delta\lambda_j$ is so big just after the first outer iteration, that all the muscles immediately explode across the scene. Tuning the stiffnesses in these large magnitudes to work well together a produce the desired result proved to be next to impossible during the development.

Therefore, a simplification was made to represent the time difference **$\Delta$t equal to one** in the context of projecting distance and dihedral angle constraints. Then, a combination of distance and dihedral angle stiffness was found ($1e^2$ and $1e^8$, respectively) which produced the expected behaviour of the muscles. This also simplified the choosing of the parameter $\beta$ of Rayleigh's dissipation potential constraint regularisation term, as it no longer had to be chosen in the terms of e.g. $1e^{-25}$, rather it was found to work the best with the value of $\beta = 0.01$, while showing a positive impact on the results in damping the constraints who's vertices moved too fast.

It is important to note, that this simulation time step $\Delta t$ simplification **only affects the projection of these two types of constraints** and is **completely independent from the real-time step $\Delta t$ used in the computation of vertex velocities and applied force displacements**, if fine-tuned properly. There may not be a direct correspondence of the stiffnesses to the mechanical properties of the parts of the muscles, but this simplification can be easily retracted to find the parameters in their realistic ranges.

# Results 7

Firstly, the four muscles of the *glutei maximus* and *medius*, the *iliacus*, and the *adductor brevis* are verified. These muscles have been chosen because they are the initially available muscle models of this project and also to isolate the results for better comparison with the results provided in paper [KČ21]. Solution on more muscles will be presented later in section 7.2.

Later, a few more results will be shown concerning intentionally chosen special cases. in the next chapters, these results will be analysed and critically evaluated.

## 7.1 Verification of solution

The four muscles are presented during the motions of hip flexion, extension, abduction and internal rotation in appendix Figures A.5-A.14 from various views. The views have been chosen to make the pronounced behaviour of the muscles visible under specific movements. Each figure has five sub-figures, wherein each of them in the down-left corner, a number in red is shown, which is the simulation step during which the screenshot was made. In every sub-figure, the **simulation step minus one is the angle of the motion in degrees**. This is due to the indexing of the steps from one. Three hand-picked snapshots under the most interesting deformation results from a particular view are added to each movement verification description. Moreover, full animations (containing all simulation steps) for each of these verification motions are available in the electronic attachment of this thesis in the sub-folder `Results\Animations` in the **.gif** format. The reader is strongly advised to observe the animations for a more complete understanding of the results.

The **XML** OpenSim plugin set-up file used was the same for all motions, except for changing the **motion_file** element to the appropriate **.mot** file. Both the **XML** configuration and the motions are available in the sub-folder `Sources\muscle-wrapping-2.0\Data\XPBDHavlicek2024` and must be extracted to the upper-level folder for the provided test solution to recognise.

The programme parameters used for these results are defined in Table B.1. They have been chosen through iterative experimental fine-tuning. And were the ones

used for every motion results. The surface fibres used for all the results are shown from the ventral and lateral views in Figures A.3 and A.4 in the appendix figures. The resolution and count of the fibres for each of the muscles are both set to **100**.

The verification in the context of this approach to muscle modelling was chosen to be done mainly from the views of

1. **visual plausibility** of the results,
2. **correspondence to the theoretical body of muscle interaction physiology**,
3. **correspondence to the results present in paper [KČ21]** (in a form of comparison),
4. **preserving of the shape** and the **volume**,
5. and existence of the **amount of muscle penetration**.

### 7.1.1 Hip flexion

Figure 7.1 shows the progression of hip flexion under degrees **20**°, **50**°, and **90**° from the ventral view. The detailed progression of the movement can be also found in the appendix in Figures A.5 and A.6 in the range of motion from **0**° to **90**° from a ventral and lateral point of view, respectively.

What is remarkable about these results is that the implemented contraction interaction solution shows an advancement in the form of the *iliacus* muscle properly following the *femur* as if it caused the movement. It is one of the prime movers during hip flexion, after all. Its contraction can be seen to start under **20**° from ventral view. Moreover, in between this and the first sub-figure in the figure, the *iliacus*

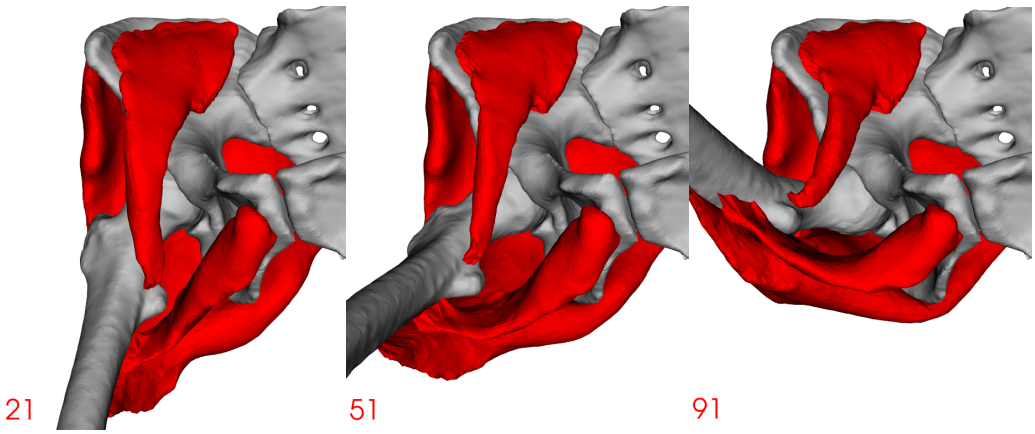


Figure 7.1: Ventral view of **20**°, **50**°, and **90**° hip flexion

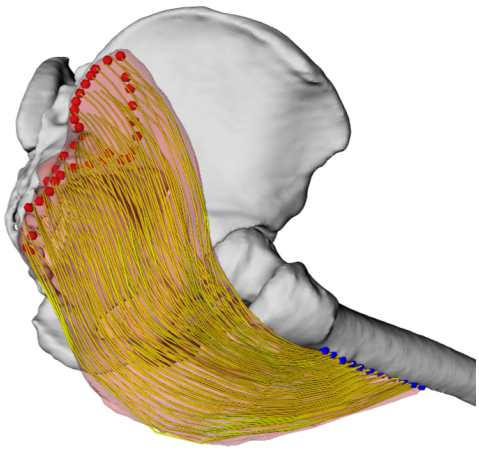

Figure 7.2: Figure (12) from the paper [KČ21], obtained through courtesy of Doc. Ing. Josef Kohout, Ph.D.

contracts progressively, as can be seen in one of the animations in the supporting materials of this work. The contraction is also supported throughout the whole body of the muscle, pulling at the attachments as if tendons were present. Because of this, the muscle deforms a bit unnaturally near its origin at the *pelvis*. It **properly keeps its original volume, as do all the muscles shown in this thesis**, as can be observed over all the appended figures.

The synergistic pair of muscles, both of the *glutei*, show no immediate signs of contraction to support the movement, probably because their contribution to the movement is very little, as only particular fibres physiologically do so. Nevertheless,

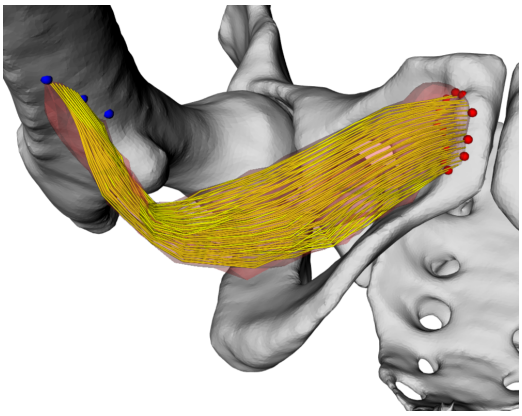

Figure 7.3: Figure (8), bottom-right corner sub-figure from the paper [KČ21], obtained through courtesy of Doc. Ing. Josef Kohout, Ph.D.

these supporting muscles seem to be deforming properly, as they both keep their shape well. The *gluteus maximus* also enters bellow the joint and *pelvis* area with its volume, which seems to be the physiologically accurate space for it to go, as opposed to staying almost rigidly in its origin position, only the part near its insertion being deformed, as can be seen in Figure *12* in the work [KČ21] or Figure 7.2. Its whole shape is deformed actively. This deformation, though, seems to be starting to cause a cavity forming under **90**° of flexion at the bottom of the belly of the *maximus* muscle (best seen from the lateral view). After closer inspection, this cavity is also present in the original shape, thus it is not clear whether or not this is a failing or a desired behaviour.

The *adductor brevis* muscle keeps its shape arguably a little bit better than the result in Figure *8* in the article or Figure 7.3, as once again, not only is the area near its insertion into *femur* deformed, but also its whole surface. But similar to the *iliacus*, near the insertion, the muscle is being unnaturally deformed by forming degenerated (elongated) triangles, this time due to the bone dragging it, as the muscle is passive in the movement.

Despite the *gluteus maximus* being pulled towards the joint, it successfully keeps its original distance to the *gluteus medius* without any penetration. Lastly, compared to Figure *13* in the paper [KČ21], the corresponding lateral view shows the *gluteus medius* behaving very similarly, while compared with Figure *14*[KČ21], the *iliacus* deformation (the ventral view) also surpasses the former solution visually.

### 7.1.2 Hip extension

Figure 7.4 shows the progression of hip extension under degrees **10**°, **30**°, and **40**° from the dorsal view. The detailed progression of the movement can be also found in the appendix in Figures A.7, A.8, and A.9 in the range of motion from **0**° to **40**° from a ventral, lateral, and dorsal point of view, respectively.

This movement shows some problematic areas of the method. Mainly the *gluteus*

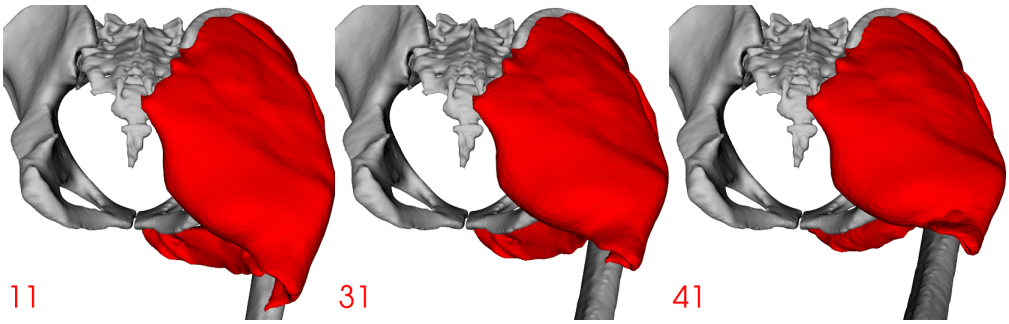


Figure 7.4: Dorsal view of **10**°, **30**°, and **40**° hip extension

*maximus* is being pushed into itself from the **20**° extension on. This is, as described in Section 2.3.2, an expected situation, as the muscle should stay relatively passive during hip extension with the extended knee (which this model has), but the results are not acceptable. This may be largely due to the shape of the muscle, where the area near its insertion into *femur* is very thin and bent even in the rest position. Preserving this shape during the extension makes the muscle eventually intersect itself, which starts to happen at approximately **20**° of extension. The question is if the muscle would stabilise towards the correct position if the simulation was left running in e.g. the maximal extension state, as the overall shape seems to be on the right path to the correct state (observe the *gluteus maximus* deformation progress from a lateral view). Nevertheless, during the movement, the muscle does not deform the way one could imagine a real muscle deform.

A possible solution to obtain better results for the *gluteus maximus* could be to represent it by a better shape, possibly inserting into the *femur* under a more medial angle, which would result in less bending in the rest-pose of the muscle. This could even potentially prevent its explosion described in Section 6.6.2 on page 93, as the attached vertices would most likely lay on the *correct* side of the mesh during the hip flexion, then no edge penetration would occur.

Another one of the seemingly problematic behaviours can be observed from the ventral view in the last simulation step. There, the *iliacus* starts to contract during its biggest elongation, and so does the *adductor brevis*. This is, however, most probably a physiologically accurate synergistic muscle interaction example, where the muscles are trying to stabilise or actuate the *pelvis* tilting (see Section 6.6.4 for associated activations), which indeed seems to be a movement typically accompanying bigger degrees of hip extension. Up to the last simulation step, these muscles seem to be deformed plausibly in shape, while correctly preserving their volumes.

Overall, the results for this kind of motion seem to be improved since e.g. the results presented in paper [Čer+23], but can not be deemed satisfactory from the deformation point of view, mainly due to the *gluteus maximus* issue.

### 7.1.3 Hip abduction

Figure 7.5 shows the progression of hip abduction under degrees **10**°, **35**°, and **45**° from the ventral view. The detailed progression of the movement can be also found in the appendix in Figures A.10, A.11 and A.12 in the range of motion from **0**° to **45**° from a ventral, lateral, and dorsal point of view, respectively.

The majority of muscle deformations of this motion yield a satisfactory result. For example, the *gluteus medius* seems to be the most pro-active muscle, while being most contracted, visibly, at approximately **35**° of adduction (see simulation step **36** from a lateral view), and then relaxing slightly at the maximal adduction (the

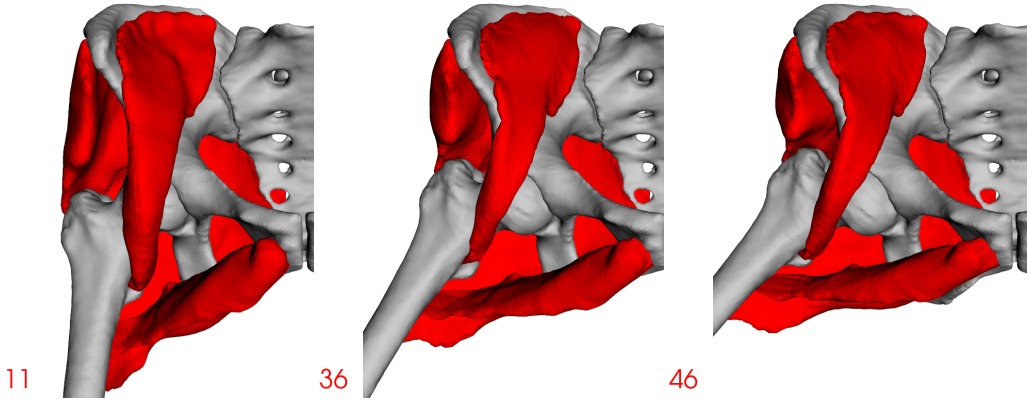


Figure 7.5: Ventral view of **10**°, **35**°, and **45**° hip abduction

following sub-figure in the same view). This follows the theory of this muscle's role in the movement precisely, which has been described in Section 2.3.3 on page 15. Another example is the *adductor brevis* seen from the ventral view, which seems to be correctly stretched across the space while keeping a plausible composure. Having said that, anomalies near its attachments are once again visible, possibly due to the distances of vertices being well-kept,

Additionally, the *iliacus* synergises the movement interactively to support the *gluteus medius* in raising the *femur*. Finally, the *gluteus maximus* seen during deformation from the dorsal view, preserves its shape well, while effectively moving with the bone once again, not just its vertices near the insertion area.

### 7.1.4 Hip internal rotation

Figure 7.6 shows the progression of hip internal rotation under degrees **15**°, **45**°, and **60**° from the Ventral-medial view. The detailed progression of the movement can be also found in the appendix in Figures A.13 and A.14 in the range of motion from **0**° to **35**° from a ventral-medial and lateral view, respectively. The ventral-medial view shows the muscles acting overall correctly, except for the maximal rotation simulation step, where the *attachment-pulling* anomalies are once more visible, this time at the insertion of *gluteus medius* into *femur*.

On the other hand, physiological accuracy can be observed. The *adductor brevis* is one of the muscles running anterior to the *femoral* vertical axis, as described in Section 2.3.6 on page 17 and as such, should, and indeed does, contract during this movement, as can be seen in its details from the ventral-medial point of view.

However, in the lateral view sub-figures, a rapture between the *glutei* is apparent. The *gluteus medius* seems to be correctly involved in the movement, as its shape continuously follows the rotation, indicating its physiological role as one of the prime movers. Whereas the *gluteus maximus* is considerably left behind below. This

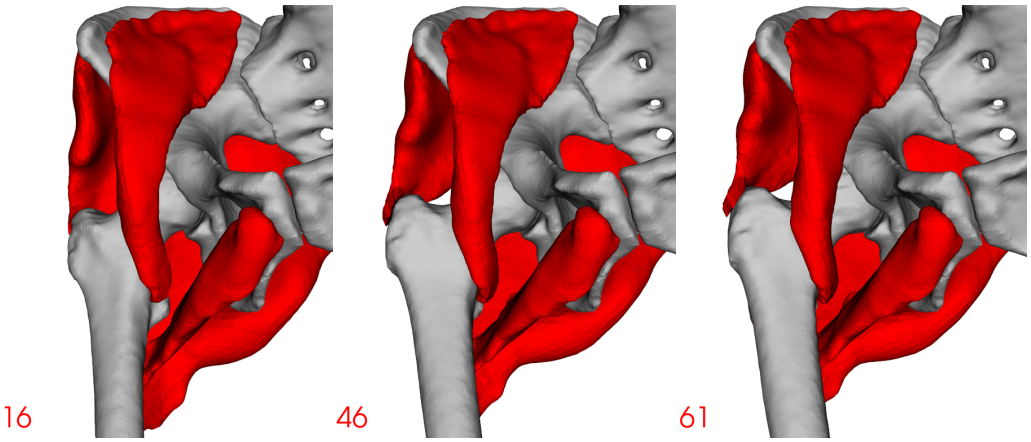


Figure 7.6: Ventral-medial view of **15**°, **45**°, and **60**° hip internal rotation

causes a rupture between the muscles to form, which most probably should not be there from the physiological point of view. Perhaps, this unsightly rapture indicates a need for distance constraints between the muscles.

## 7.2 Added muscles

After adding all the muscles shown in Figures A.1 and A.2, immediate fine-tuning of the scene had to be made for the case of hip flexion, as a few of the muscles started exploding immediately under the same parameters (Table B.1) which were used for verification of the four basic muscles. Each muscle had **100** inner fibres of resolution **100** generated.

These muscles started to explode in the first iterations, since as opposed to the truthful rest-position shown in Figure A.1, the simulation starts with the *femur* bone under a slightly different rotation. This causes the bone to appear colliding with the muscles, which is usually not a problem for the muscles near the *pelvis*, but at the distal parts of the *femur*, this slight change of rotation introduces a large change in the bone's end position. Therefore, it makes sense, that the exploding muscles were indeed the ones attached to the distal part of the *femur*, or even under the knee. The significant difference in spatial relations of the muscles and the *femur* is illustrated in Figure 7.7 compared with the rest-pose muscles.

To avoid the initial explosion, **the stiffness of distance constraints was raised to the value of** $1e^5$, resulting in the progression depicted in Figure 7.8. In the figure, the majority of the muscles running near the knee can be seen hanging lowly and unrealistically. With these muscles, it is debatable, whether they even are muscles still in that area and not mostly just tendons and should therefore behave like tendons. Their *true* rest-pose composure around the knee (Figure A.2) breaks up

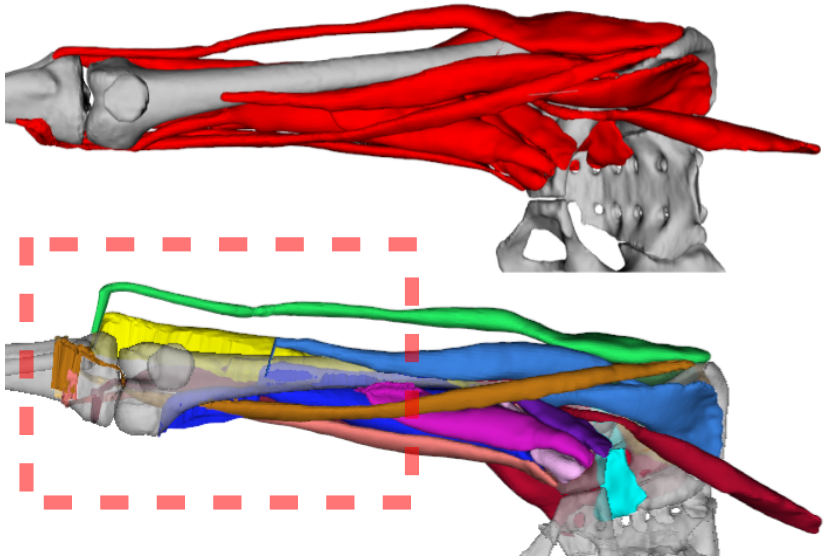

Figure 7.7: Difference between *true* rest-pose of the muscles versus the simulation start causing collisions, with a problematic area highlighted in red dashed rectangle

completely even before the simulation even starts (due to the *femur* transformation). Next, the up-most red *psoas* muscle is visibly on the verge of an explosion under the limit flexion of **90**° (image on the right in Figure 7.8). Moreover, in the figure, the

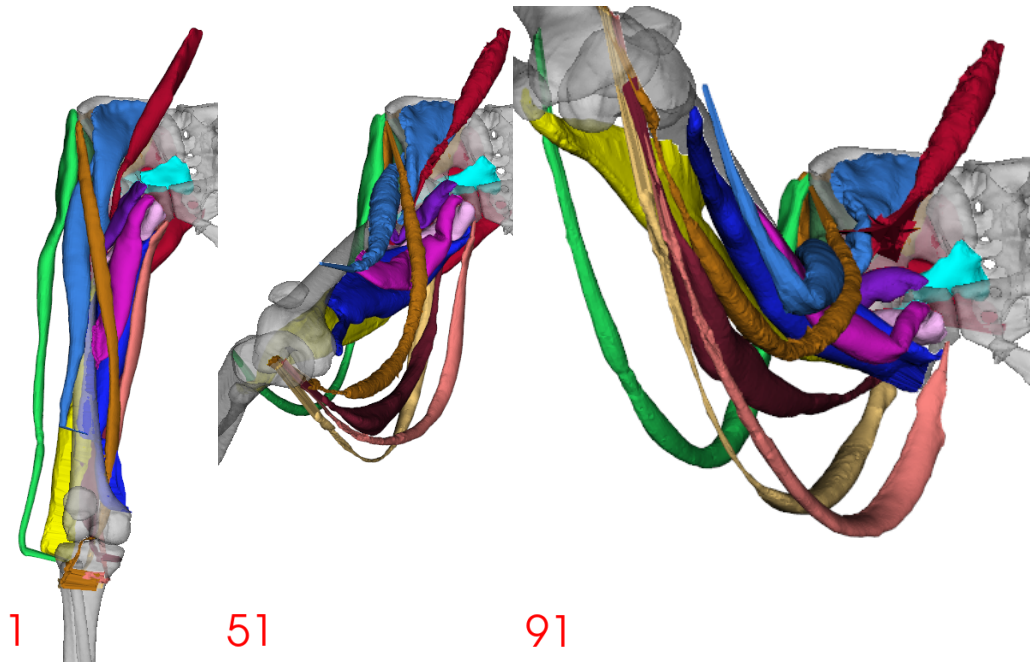


Figure 7.8: Ventral view of **0**°, **50**°, and **90**° hip flexion of all muscles with higher distance constraint stiffness

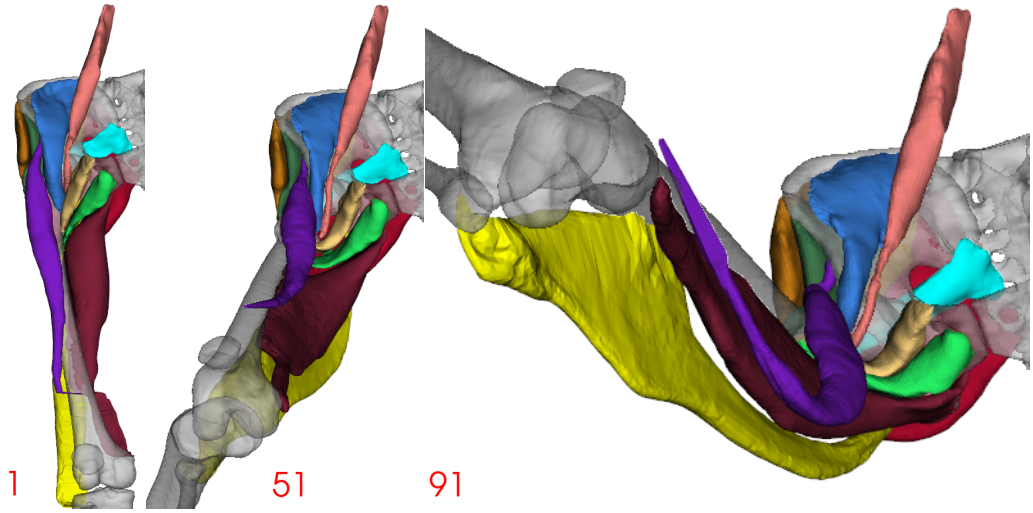


Figure 7.9: Ventral view of **0**°, **50**°, and **90**° hip flexion of many muscles with original parameters

*gluteus maximus* can be seen in red in the background for the first two snapshots but vanishes in the right-most one. This is because of the previously discussed problem of muscle explosion (see Section 6.6.2 on page 93). This is mostly explainable by the higher distance stiffness parameter, which constrains the *gluteus maximus* more, resulting in the sooner emergence of the penetrating edges.

The solution to the low-hanging muscles might be to model them heterogeneously with the incorporation of some level of rigidity near the knee attachment areas based on their underlying anatomy (to decide which parts should be more rigid). These muscles will no longer be accounted for in the following results due to the obvious violations of their non-trivial shapes.

The solution of the *psoas* exploding was solved by the observation, that it is contracted too harshly. Lowering the number of inner fibres from **100** to just **16** shows the psoas working properly during the hip flexion in Figure 7.9 throughout the simulation. With the original parameters defined in Table B.1 and the removal of the problematic muscles, the remaining muscles seem to deform somewhat adequately. The most problematic is the *rectus femoris*, which keeps up with the bone movement well (as it is being pushed upwards) until approximately **50**° of flexion, where it starts to slide down to the side (centre image in Figure 7.9). Moreover, the *biceps femoris* is also not following the movement of the bone well enough, even though it is attached at many positions near the knee area. But what this scenario shows the best, is that the collisions between the muscles are avoided very well. The reader is invited to see the attached electronic sources, where an animation of a detailed view of these muscles avoiding penetrations is available on the path `Results\Animations\All-muscles\muscle_collision_handling_detail.gif`.

## 7.3 Critical evaluation & Discussion

A part of the critical evaluation has already been done during the validation of the results, as many interesting situations were observed, which could be hardly put into any other context than that of them being compared with the theoretical physiology (which the method is trying to reach) and practical results from the canonical PBD article [KČ21] (upon which the method is based). Nevertheless, a more general evaluation of the method is in place, as many problematic topics have also been touched upon during the description of the development of this model.

### 7.3.1 Drawbacks

The main problematic areas of this proposed solution seem to be stemming from

1. a too naive of a detailed surface fibre extraction method,
2. and a possible over-constriction of the muscles.

These two sources of problems often interact with each other. For example, the less numerous the surface fibres, the less chance of the muscle being over-constrained during contraction, e.g. in the case of the *psoas* muscle (Figure 7.8 on the right) up to the point of the muscle explosion initiation. A very similar situation happens in the case of the *gluteus maximus* muscle, which seems to be over-constrained in the sense that it is keeping its shape almost too well near its insertion into *femur*, where it should perhaps allow more shape deformation, also eventually leading up to a muscle explosion.

Both cases of registered explosions happen at rather extreme cases, considering **100** surface fibres for such a thin muscle as the *psoas*, or the occurrence of such badly degenerated triangles of the *gluteus maximus*, penetrating a bone quite severely as has been shown in the immediate reflection on the problem.

In the case of the *psoas*, it can be argued that this explosion is caused by the contraction of the surface fibres, which, for example, run chaotically from side to side around this very thin muscle area, and when contracted at the same time, choke the muscle so much, that the dihedral angle constraints become unstable and start generating great costs, which result in greater internal forces generated, which often produces the dihedral angle violations again, repeating the cycle until the area explodes.

The situation seems to be more complicated concerning the near-attachment area of the muscles. There are two main counter-acting effects at play. As the verification of the four basic muscles has shown, the muscles preserve their shapes and volumes very well while also expressing a high level of elasticity (for example

they generally follow the moving bone closely with their shape), which is a desired property of the system.

But at the same time, due to this property, the muscles often showcase an anomaly of degenerated triangles forming near the attachments, mostly when the bone pulls on them, but also in other cases, e.g. when the muscle contracts a lot. Sometimes, these degenerated triangles enter the bone with their edges, as no edge collision handling is being done, and eventually, such internal forces are generated, that the muscle can tunnel through the bone suddenly (in just a few simulation steps) in an explosive manner.

One situation is probably solvable by carefully designing unique surface fibres per muscle, e.g. by hand or by the change of the parameters (fibres count and their resolution). The other situation is perhaps solvable by avoiding edge collisions with the bones.

However, adding new parameters to the system may start to make it too complex, so it would be best for these problems to be solved by tuning the parameters already present. This leads to the third problem, which is the **sensitivity to parameters**.

In other words, there does not seem to be a fit-for-all global configuration for the method as of now, such that it behaves correctly in all cases, apart from fine-tuning the parameters (maybe even for individual muscles).

One of the main battles happening during the muscle explosion is the argument of the constraints. For example, making the distance constraint stiffer will prevent some of the explosions where no bone is present in the near, but it will also make the explosions likely to happen sooner near the attachments.

### 7.3.2 Merits

While the majority of analysis of the results has so far been done on the drawbacks of the model, it also shows many promising results. The main positive properties of the system are

1. the successful realization of the intermuscular interactions
    - through the avoidance of muscle penetrations,
    - and active, synergistic, physiologically corresponding contractions of muscles,
2. the resolution of *iliacus* unnatural bending during hip flexion,
3. the introduction of simulation-time independent constraint behaviour

Incorporating more muscles may have not put the deformation method on a pedestal, as it was developed mainly for the muscles verified, but instead, the results

on these muscles have showcased surprisingly well-avoided penetrations of the muscles, even in a scene, where many pairs of muscles come to contact proximity at once, even from an unexpected direction (see the electronic attachment of this thesis at `Results\Animations\All-muscles\muscle_collision_handling_detail.gif`).

In most of the presented results, visible showcases of the physiologically accurate behaviour of the muscles can be observed. The resulting deformations seem surprisingly correct not only in the literature-corresponding timing of contraction but also in visually plausible contraction expression, where the muscle when it is supposed to contract, its whole surface contracts also, often forming visible bellies, at the places where one would expect the volume of the muscle to accumulate. An exemplary muscle regarding this topic is the *gluteus medius* during hip abduction. Based on OpenSim's SOT activations, which are in theory just estimates based on very rough muscle approximations by polylines, the activations of this muscle during hip abduction seem to behave as if cut out from the [KOA19] physiology book. A part of these results may be attributed to the anisotropy distance constraint modification concept brought forward in the research of the original PBD method [KČ21].

Furthermore, the historically problematic behaviour of the *iliacus* muscle during hip flexion seems to be resolved by the proposed method. The muscle, either being caught in between the hip joint cavity, as results in paper [KČ21] present, or being unnaturally bent at its insertion into *femur* when using the Discregrid for muscle-bone collision handling, neither of these problems is present.

The lively scenes, where almost no muscle stays rigidly idle, are also possible thanks to the implemented XPBD method, which ensures the muscles stay dynamic across the movement simulation, consistently. Adding to that, the efficiency of the implementation allows for higher solver iteration counts, giving the constraints enough room for expression, making the muscles preserve their shape well, as well as their volume and vertex distances.

The final aspect to evaluate is how well the reconstructed inner fibres change their lengths according to the expected deformation, and also compared to the original PBD approach [KČ21].

### 7.3.3 Fibre lengths

The fibre lengths have all been measured on the same muscle configuration, which is to use **100** fibres per muscle, which are each divided into **15** segments while parametrised by the deformed surface through **Mean Value Interpolation** [KČ21]. As opposed to the results shown, where each fibre is split into **100** segments, to fairly compare the methods, even though the XPBD method uses the number of segments to do the deformation itself and is dependent on it.

Figures 7.10 and 7.11 each show a pair of muscles, where the fibre lengths are measured. Figure 7.10 shows in the first row the *gluteus maximus* and in the second row the *gluteus medius*, while Figure 7.11 shows the *adductor brevis* in the first row, while the second row contains the fibre lengths of the *iliacus*. The reconstructed fibre lengths are estimated for each simulation step of hip flexion from **0**° to **90**°, advancing the *femur* rotation by **1**° per simulation step. An important difference is that the PBD solver used here uses 3 internal solver iterations, where it provides plausible results and the value is set as the default for the method, whereas with the XPBD method, utilising the consistency of constraints during high iteration count, will be left at **70** internal iterations, as this is the (so far) default for the XPBD.

What is notable about the right column in Figure 7.10 are the fibre lengths elongation in the last simulation steps near the degree of **90**. There, the broadly discussed muscle explosion starts to happen, as the vertices start to tunnel through the *femoral* bone near the attachments. The results for the XPBD method seem to correspond well to the results of the PBD, where as noted by the authors of the paper [KČ21], the *gluteus maximus* behaves expectedly, as mostly its deep fibres lengthen. These

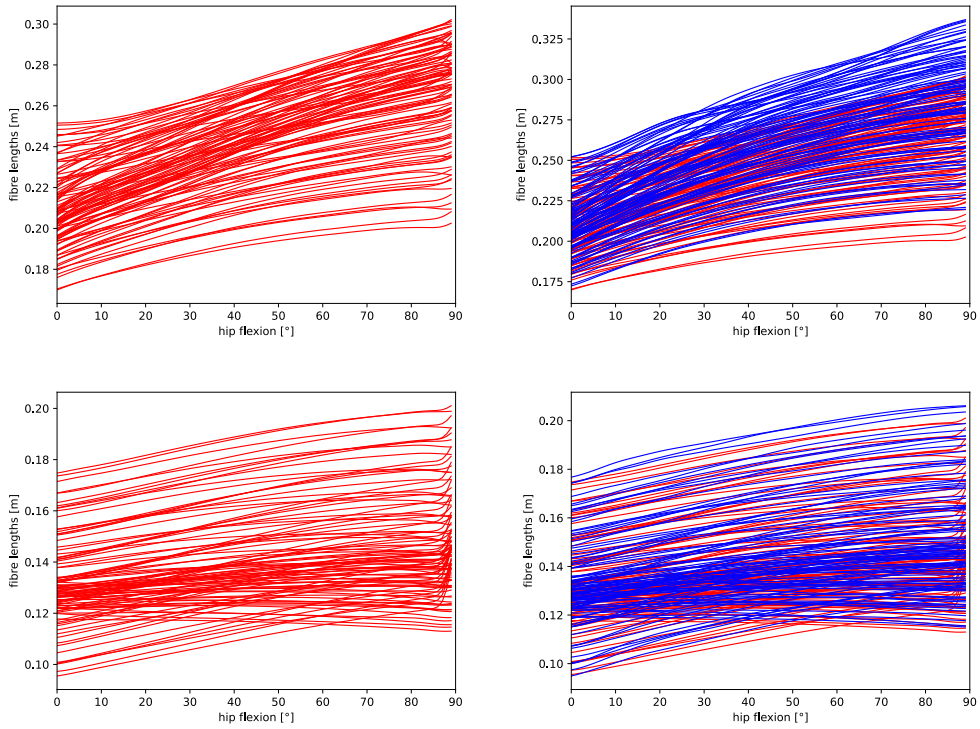


Figure 7.10: *Gluteus maximus* (top row) and *gluteus medius* (bottom row) fibre lengths during simulation of hip flexion, left column is the XPBD method results in red, right column contains the same results for XPBD in red with PBD method results in blue

deep fibres can be identified by the group of fibres lengthening more than the others. At the same time, the whole muscle lengthens. Further, as the authors note, the expected behaviour with *gluteus medius* is that only its surface fibres extend, while in the plot, a group of fibres is also shortening, these are probably the deep fibres of the *gluteus medius*. Compared with the results in the right column, the fibres of *gluteus maximus* extend much more sharply using the PBD method, maybe even the speed of their lengthening is rising, compared with the XPBD method. this can be due to the observation, that the XPBD method moves the whole shape of the muscle much more dynamically and also keeps the elasticity of the muscle consistent, while the PBD method, in a sense, prevents the muscle from entering into the joint area, forcing the fibres to extend more to reach the attachment. Moreover, there is a chance that the change in the lengthening speed of the methods is due to the possibility, that the PBD method exerts less elasticity than the XPBD method, while the stiffer parts of the muscle force the externally pushed parts to extend more. The fibre lengths in *gluteus medius* follow a similar trend, but using XPBD, the fibres are ever-so-slightly shorter during the whole movement, possibly because of letting the muscle move more freely and adapt to the external forces more responsively.

Both cases make sense, as the XPBD provides more iterations of solving, but the same number of iterations (70) could have not been compared with the PBD method, as it failed to complete the simulation in keeping the *gluteus maximus* in an acceptable shape using the higher number of iterations.

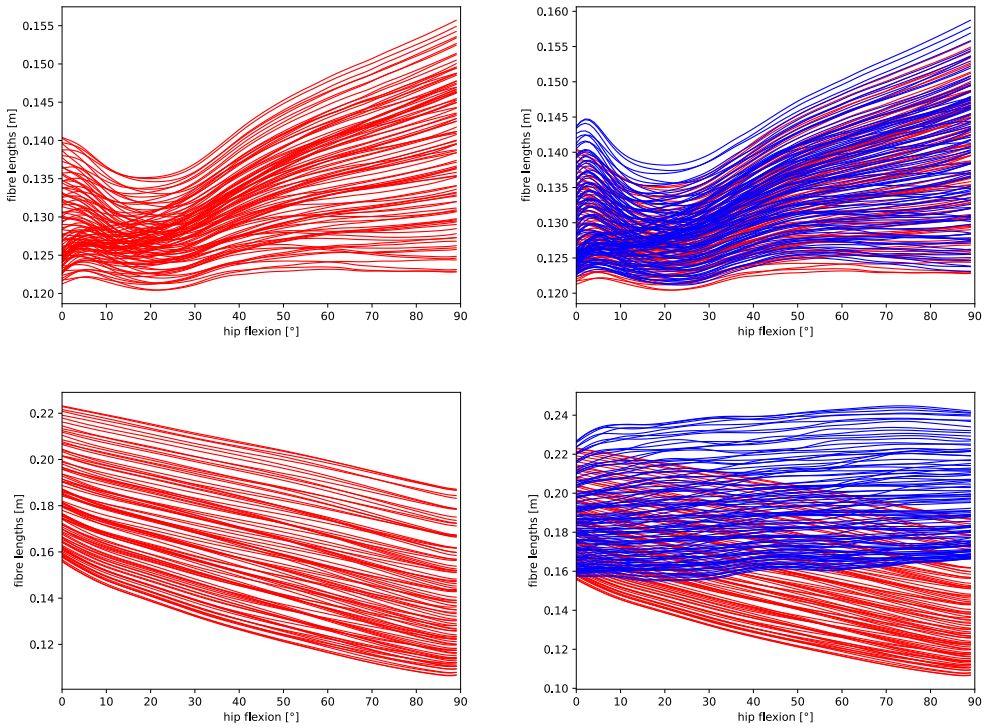


Figure 7.11: *Adductor brevis* (top row) and *iliacus* (bottom row) fibre lengths during simulation of hip flexion, left column is the XPBD method results in red, right column contains the same results for XPBD in red with PBD method results in blue

In Figure 7.11, the results of the XPBD method for the muscles *adductor brevis* (top) and *iliacus* (bottom), seem feasible. For the *adductor brevis*, this result could be explained due to its anatomical position about the *femur*. First, the muscle is picked up at the start of the motion, causing it to bend a little to face the *femur*. When the *adductor brevis* is directed perpendicularly to the *femur* (at 20°), the distance between its attachments is the lowest across the movement. From that point on, the muscle insertion is pulled upwards by the *femur*, making the muscle stretch. For the *iliacus*, the result is also expected (and corresponding to the results shown in the article, where the voxelisation collision handling is used instead of the Discregrid), as the muscle should contract continuously, being one of the two prime movers of hip flexion, although the fibres are shorter compared with the result in the original paper by approximately 1 centimetre. Compared with the current version of PBD using the same collision handling as the XPBD, a marginal difference can be observed, as the passive dragging of the *iliacus* by the PBD method, while the fibres even lengthen a little, the XPBD method effectively contracts the muscle, shortening the fibres as expected.

# Conclusion 8

A fast and efficient approach to detecting and avoiding collisions of deformable bodies, based on stochastic methods was designed and implemented. The original deformation approach [KČ21] has been refactored and extended into its successor method to overcome a well-known problem of time dependency. This extension enabled the development of a proposed model for active intermuscular interactions under arbitrary movements, maintained through physiologically-based parameter management to reflect each muscle's role in the movement.

The results were rigorously verified for visual plausibility, adherence to the physiology of muscle interaction, correspondence to the results of a similar method, and the preservation of muscle shapes and volumes. The quality of the proposed muscle penetration avoidance method was also assessed. The verification was conducted on four of the provided muscles of the *glutei maximus* and *medius*, the *adductor brevis*, and the *iliacus* during five basic leg movements of flexion, extension, abduction and internal rotation. Additionally, all identified drawbacks were analysed exhaustively, and possible solutions were discussed. Moreover, the scene was enriched by 16 new muscle models. In the end, the most relevant weaknesses and merits of the proposed method were critically evaluated, as well as the final comparison with the former method in terms of the plausibility of the inner fibres reconstructed from the deformed muscles was made. Based on this information, the assignment can be considered fulfilled.

All the implemented propositions proved to be successful in their original ambitions, surpassing the idea of the method being a mere proof of concept, as its outcomes seem to show great signs of applicability to real, bio-mechanical systems. The method still lacks a robust mechanism to do that, but advancing this research by e.g. approximating the inner muscle fibres more accurately on the surface mesh, handling muscle edge-to-bone collisions properly, or maybe even validating the outputs against real EMG data, could eventually resolve these drawbacks.

# Figures A

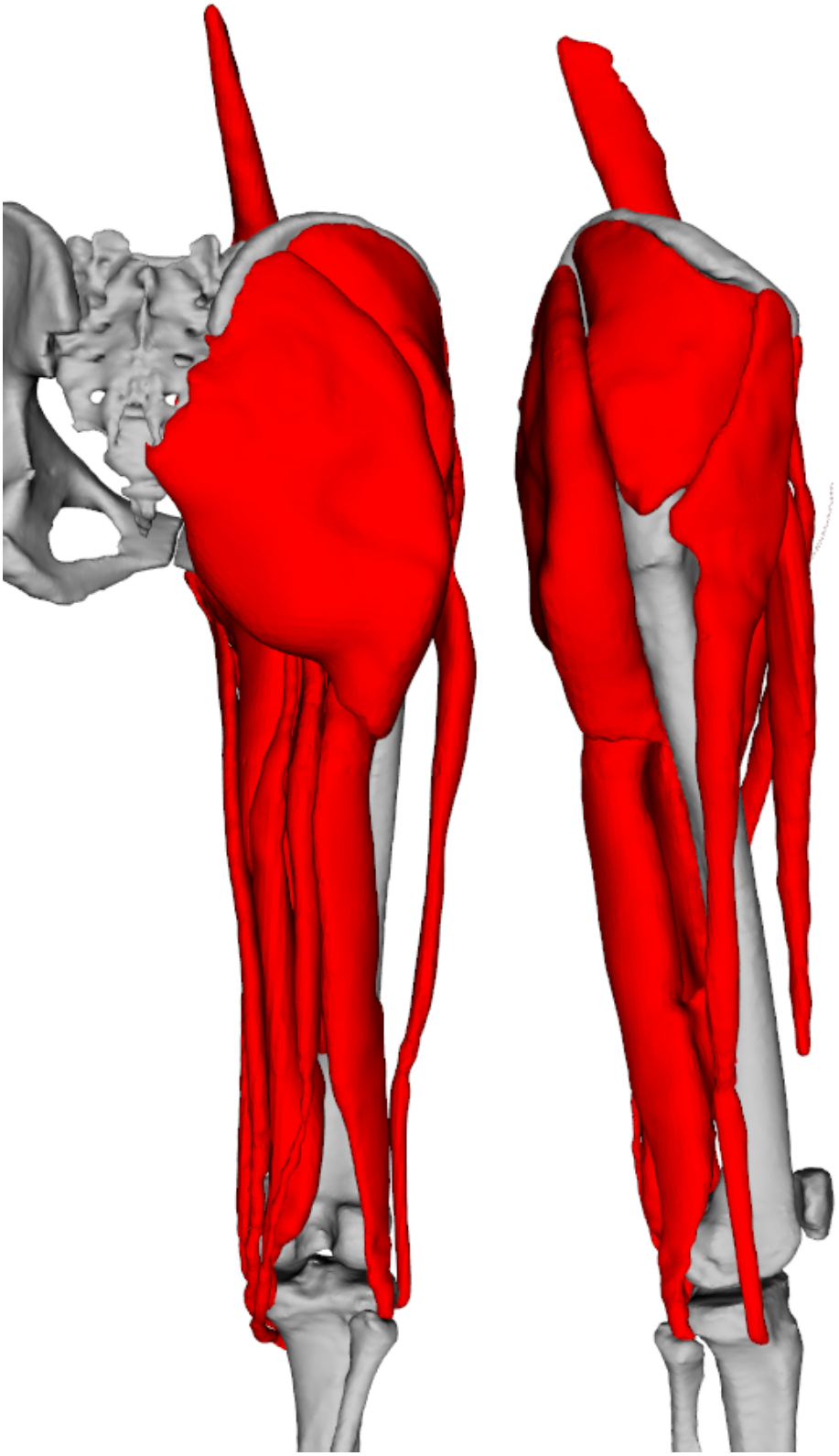

Figure A.1: Dorsal (left) and lateral (right) views of all possible muscles

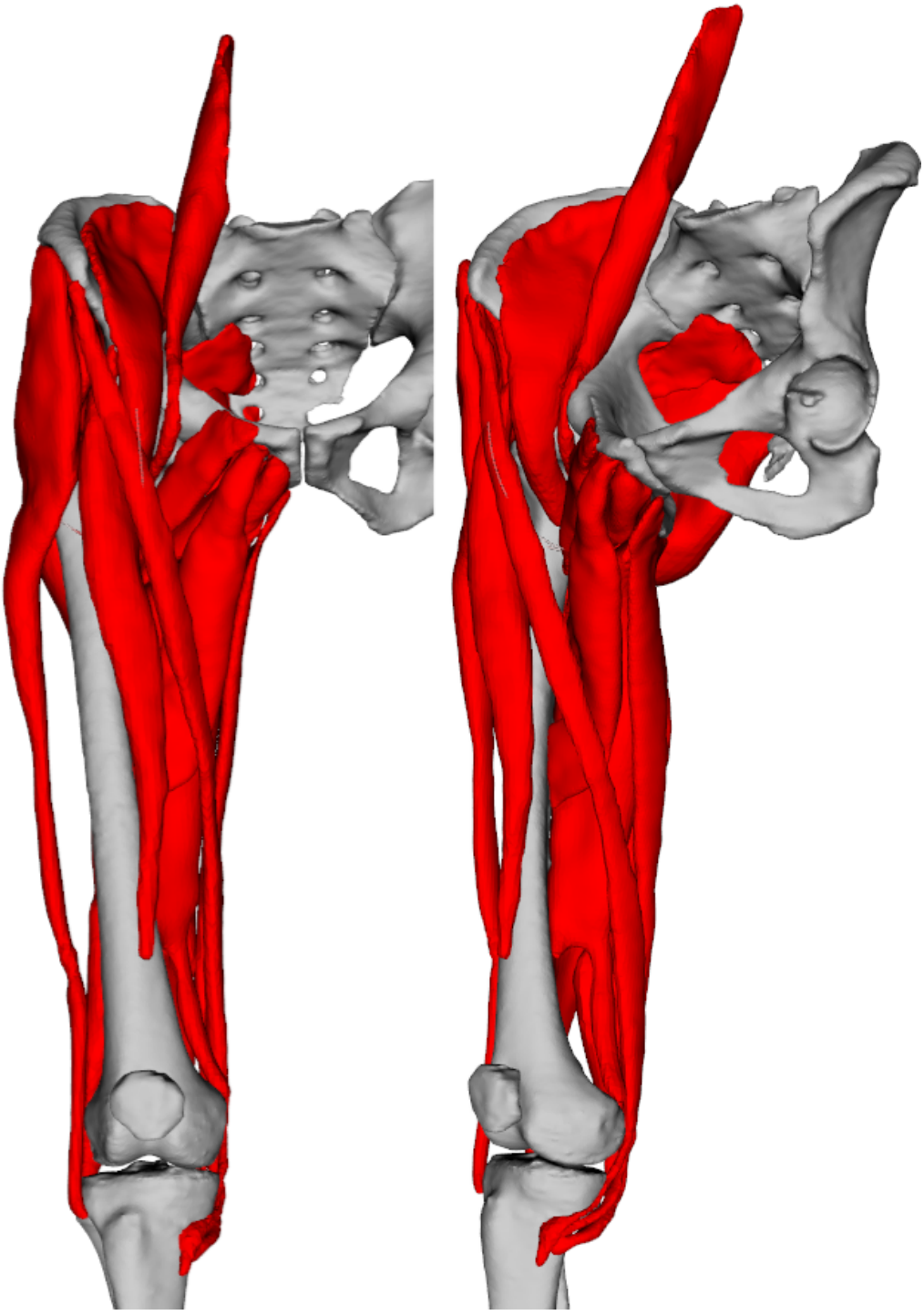

Figure A.2: Ventral (left) and medial (right) views of all possible muscles

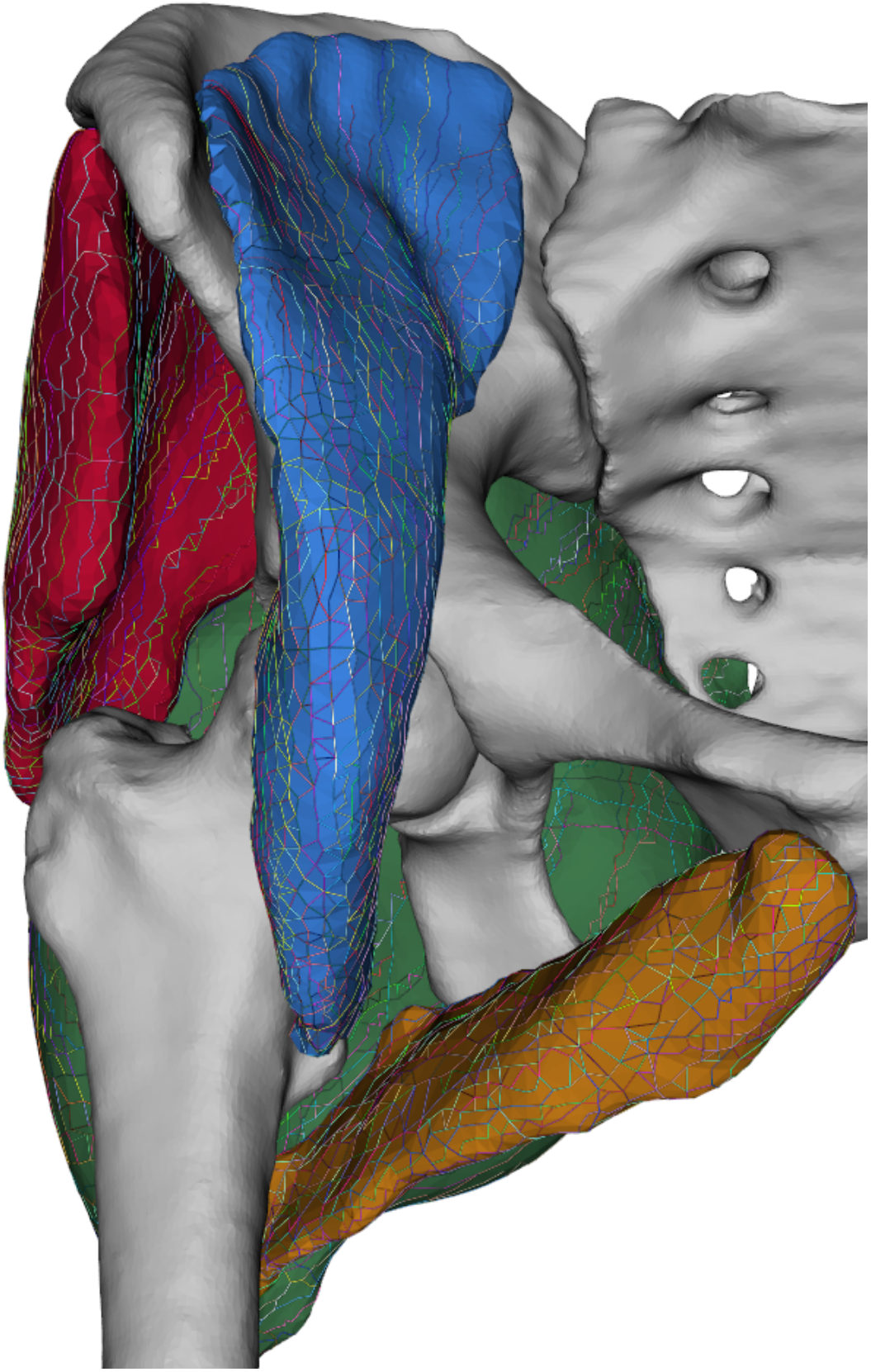

Figure A.3: Ventral view of the surface fibres used for verification

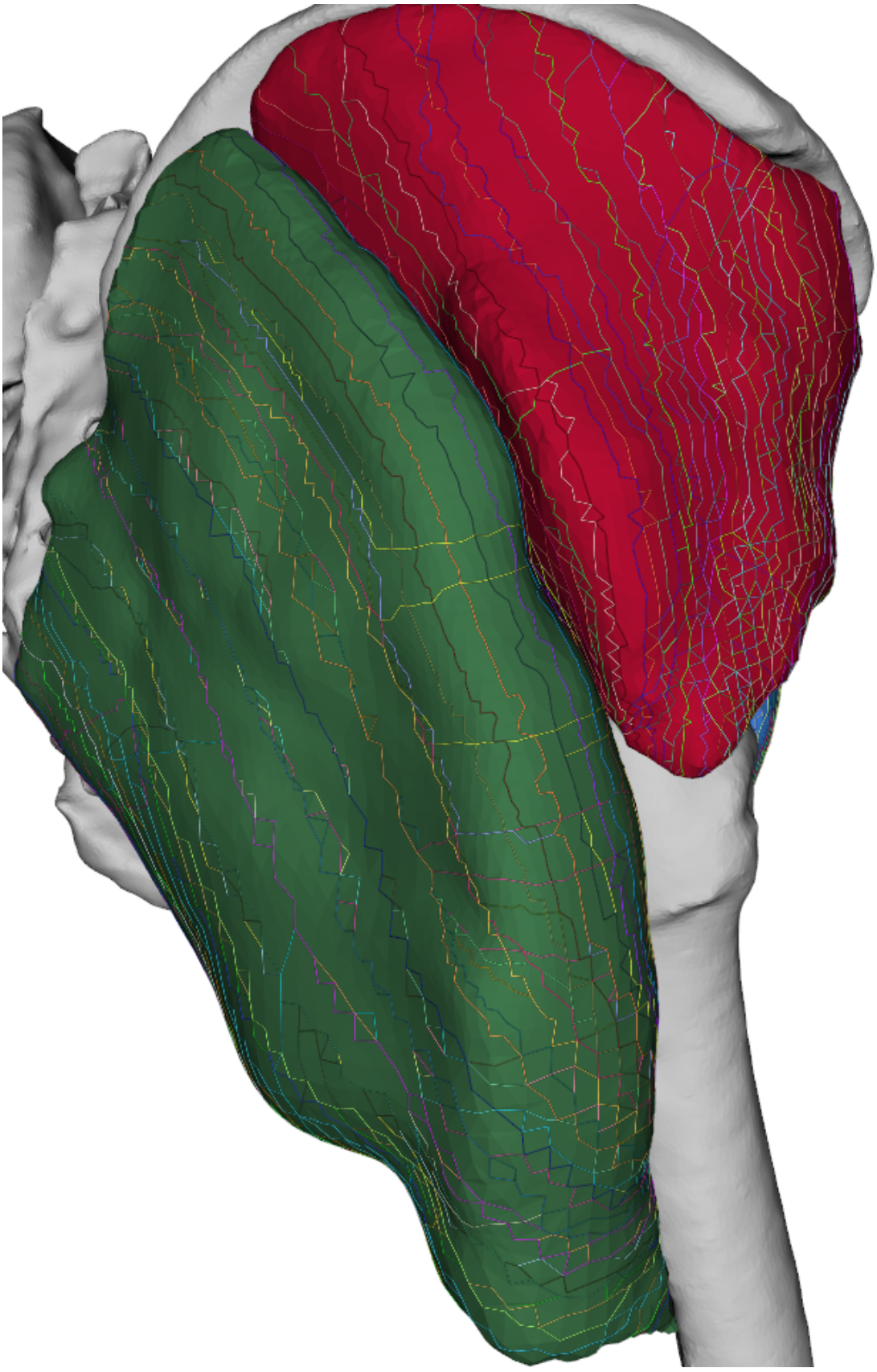

Figure A.4: Lateral view of the surface fibres used for verification

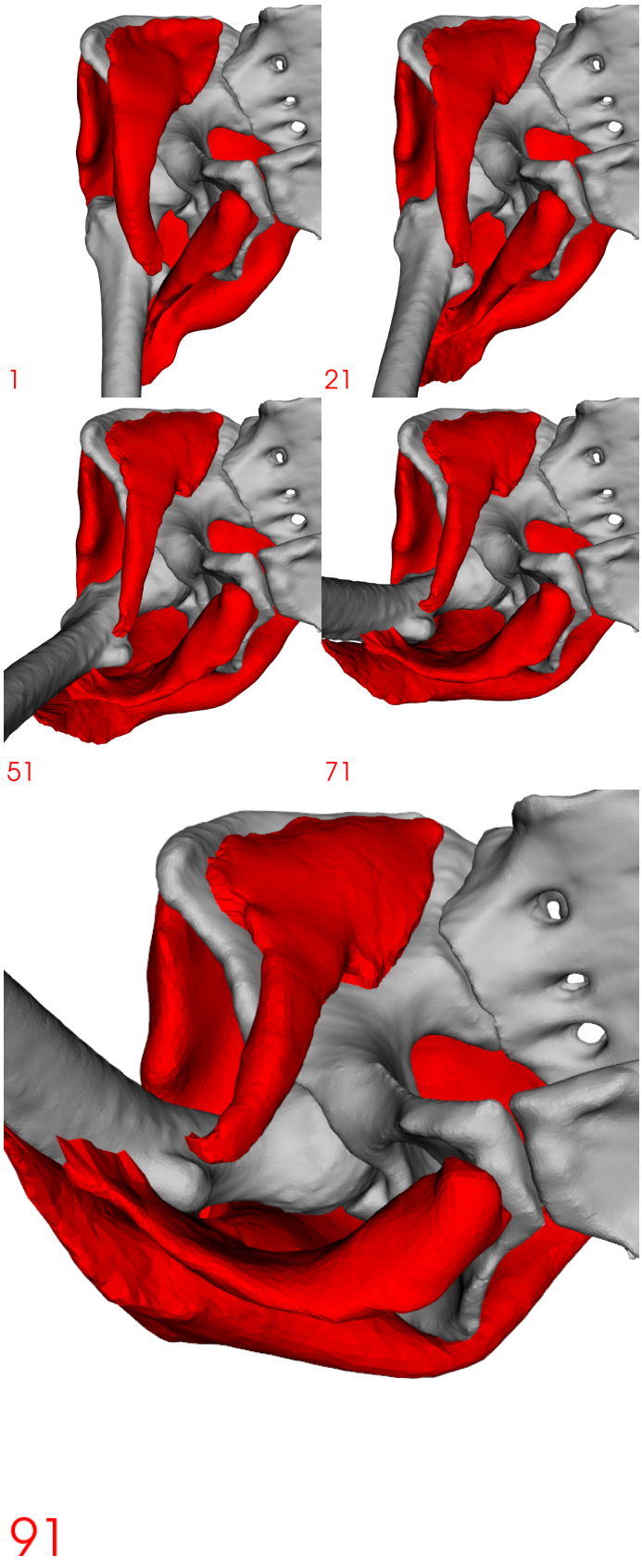


Figure A.5: Ventral view of hip flexion

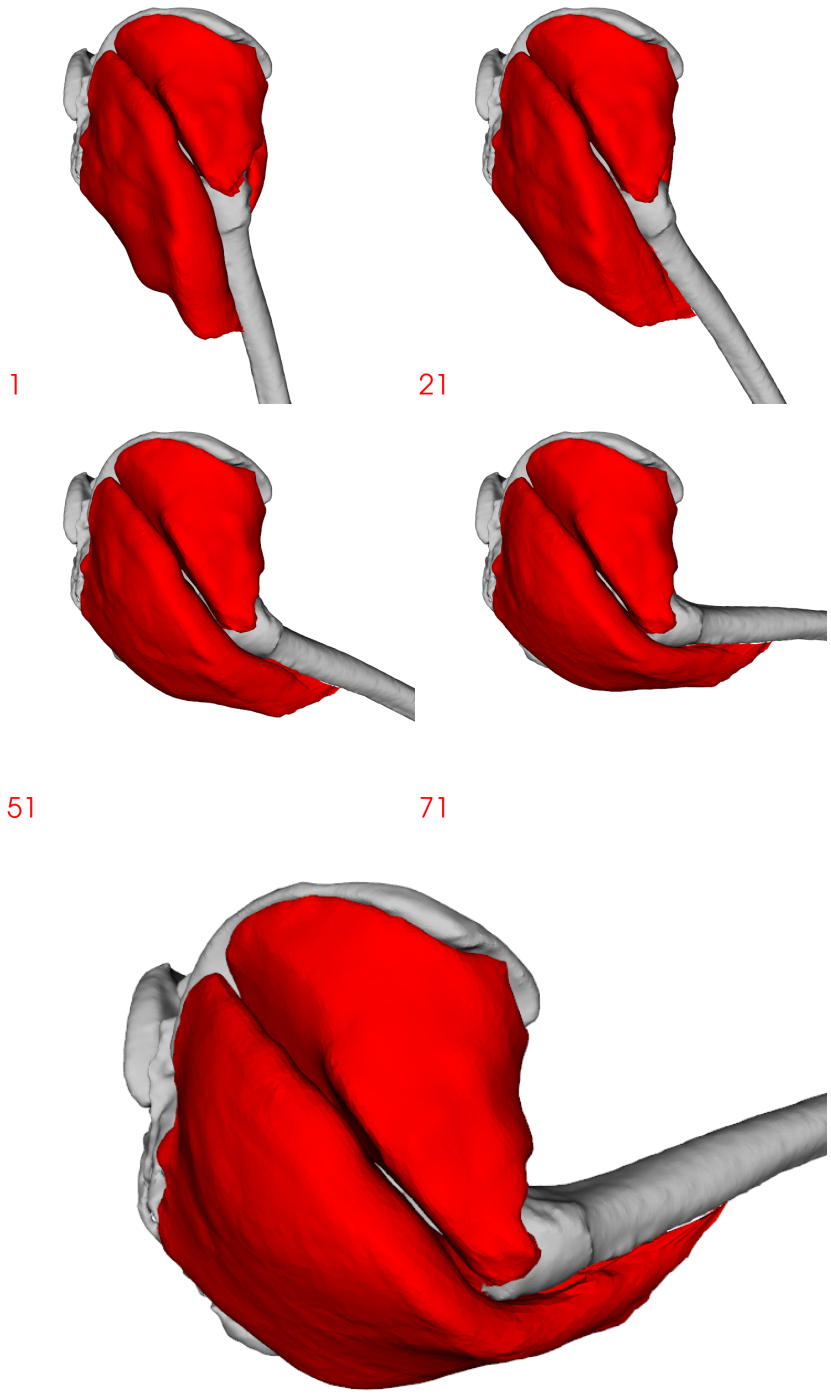


Figure A.6: Lateral view of hip flexion

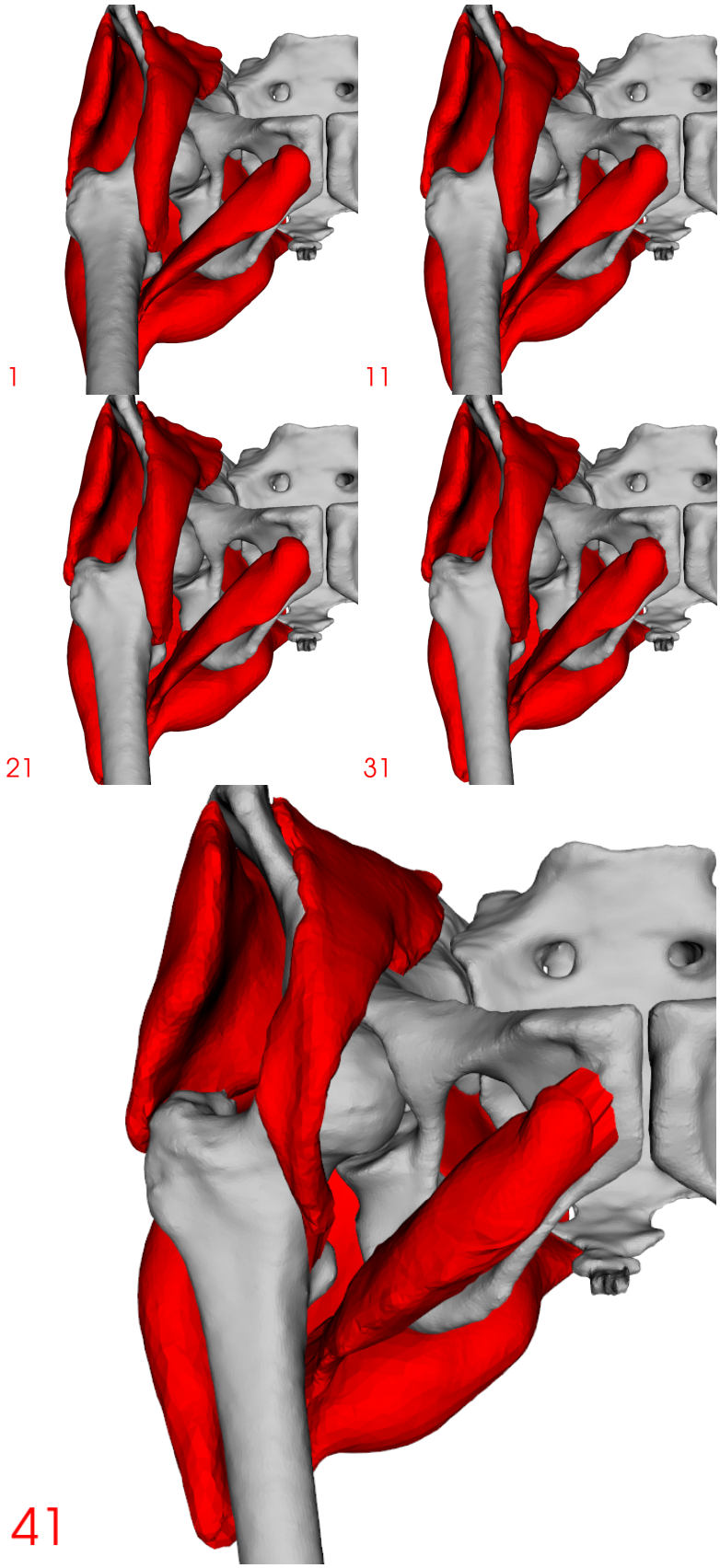


Figure A.7: Ventral view of hip extension

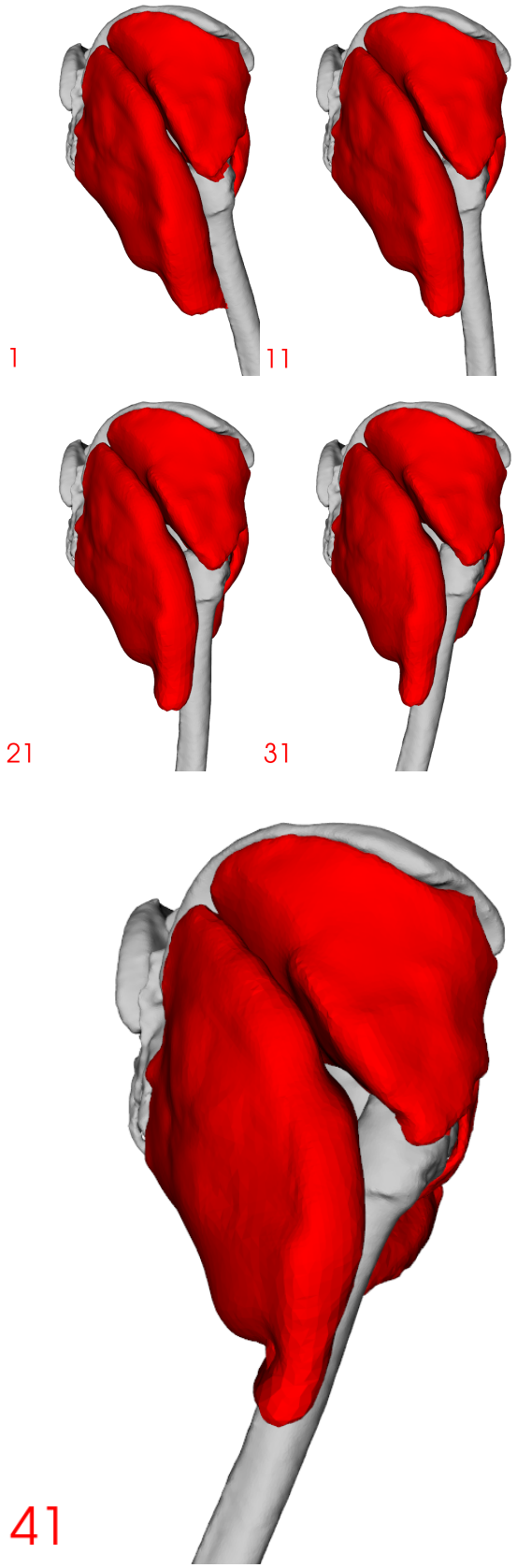


Figure A.8: Lateral view of hip extension

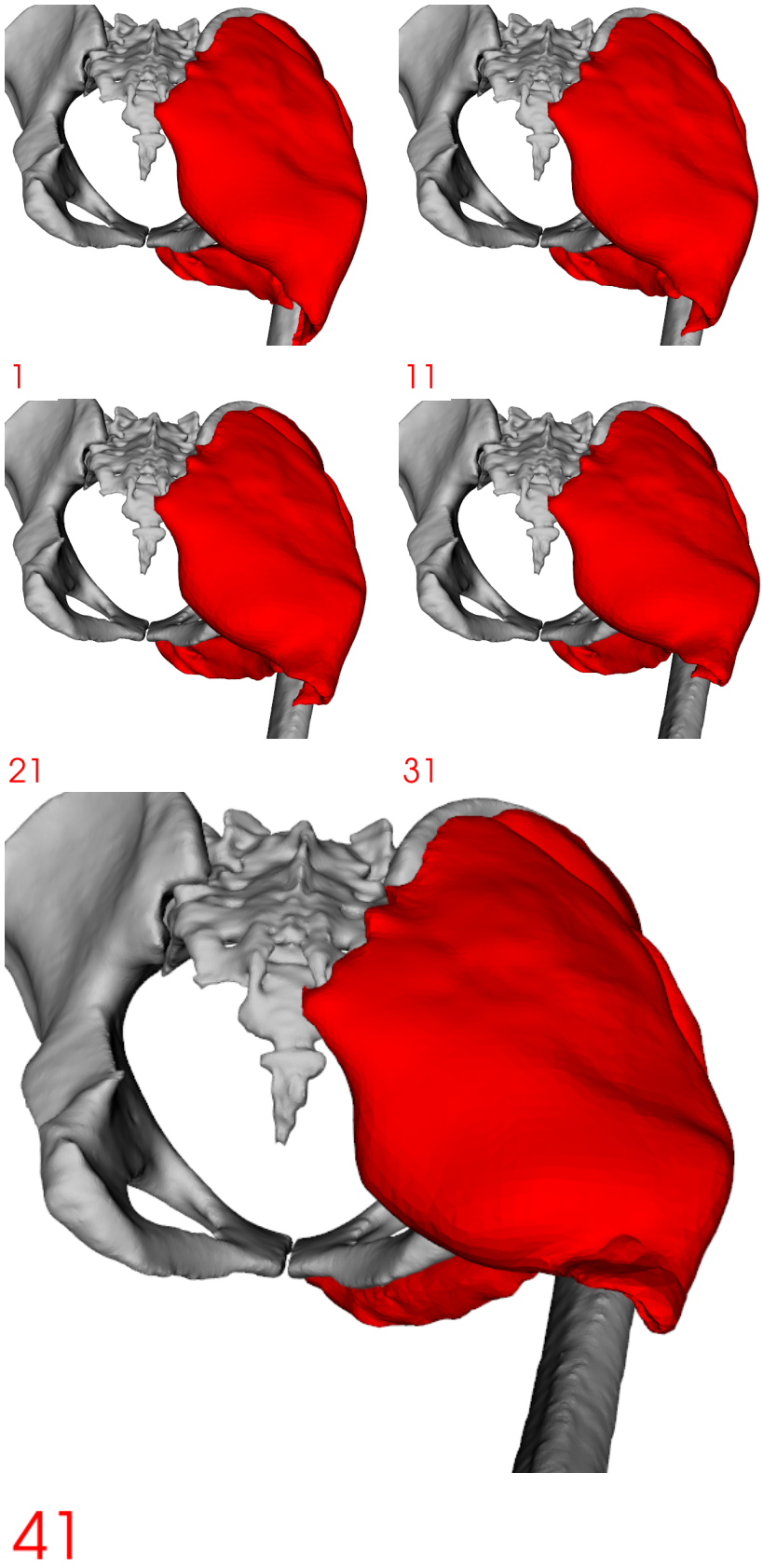


Figure A.9: Dorsal view of hip extension

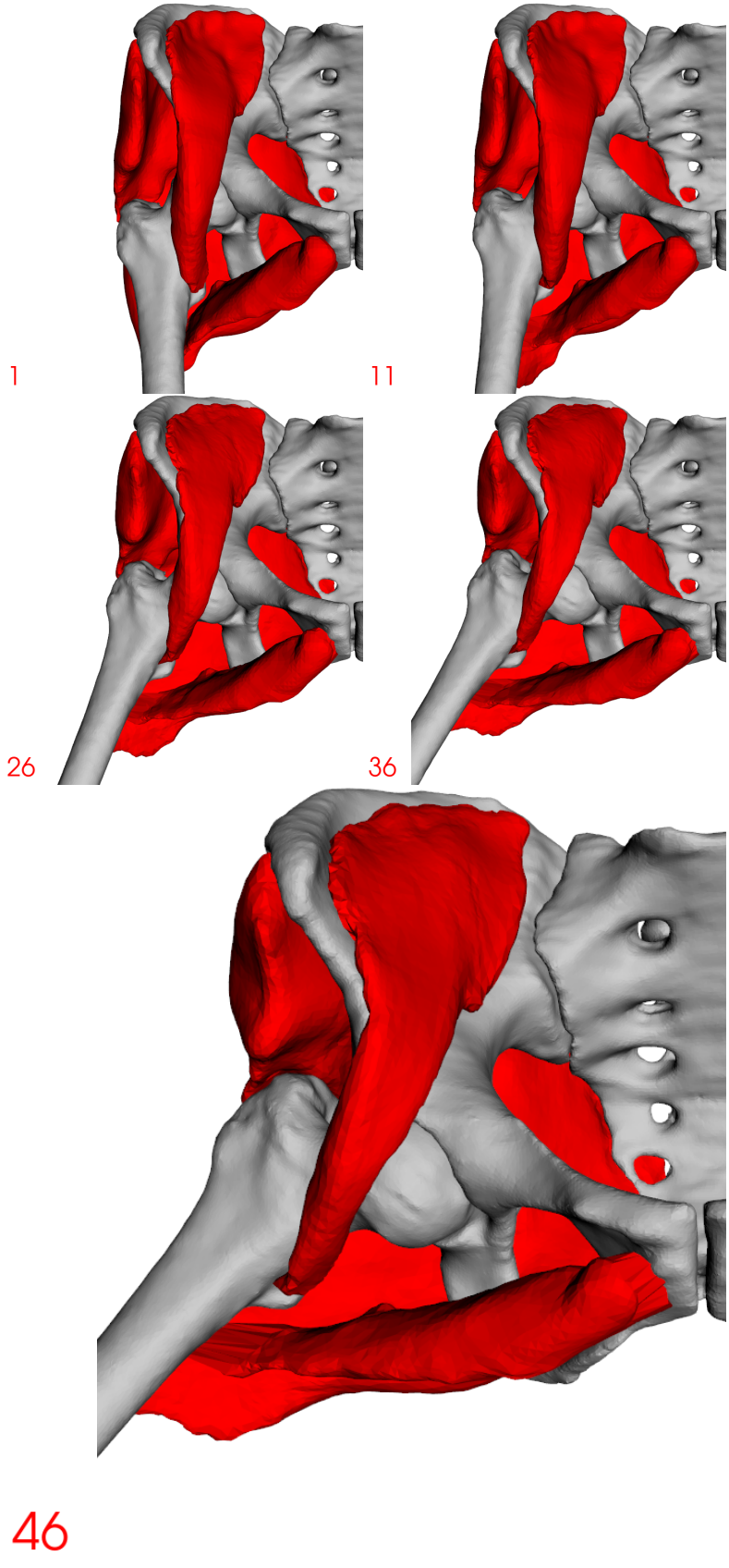


Figure A.10: Ventral view of hip adduction

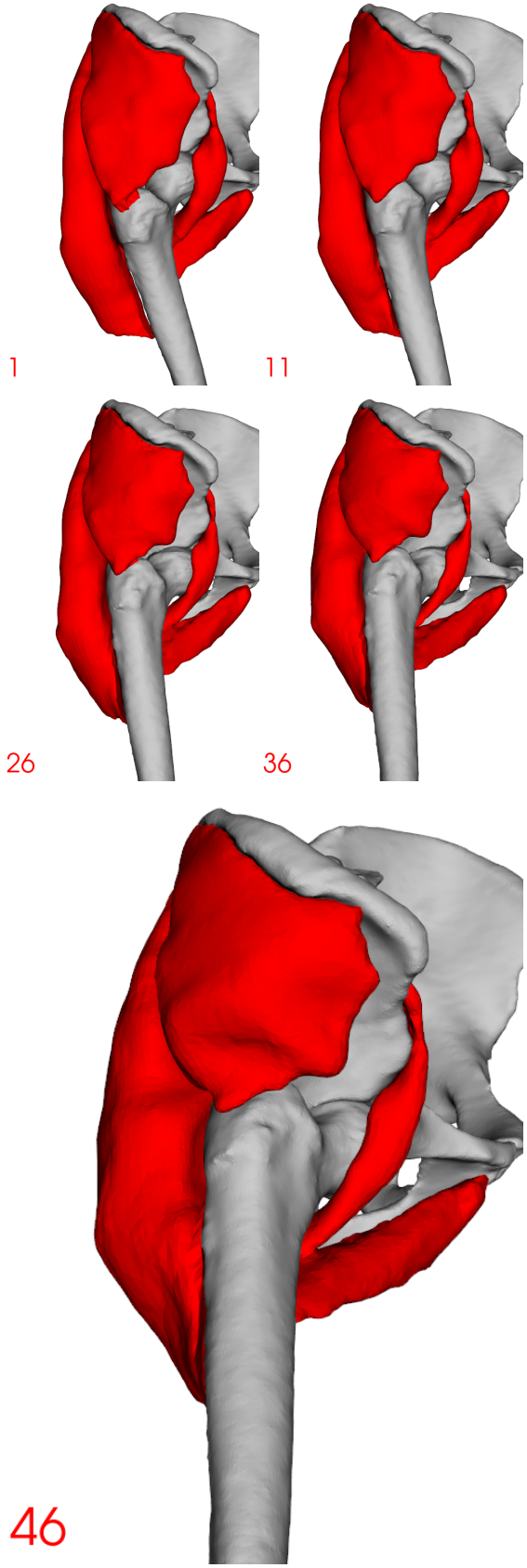


Figure A.11: Lateral view of hip adduction

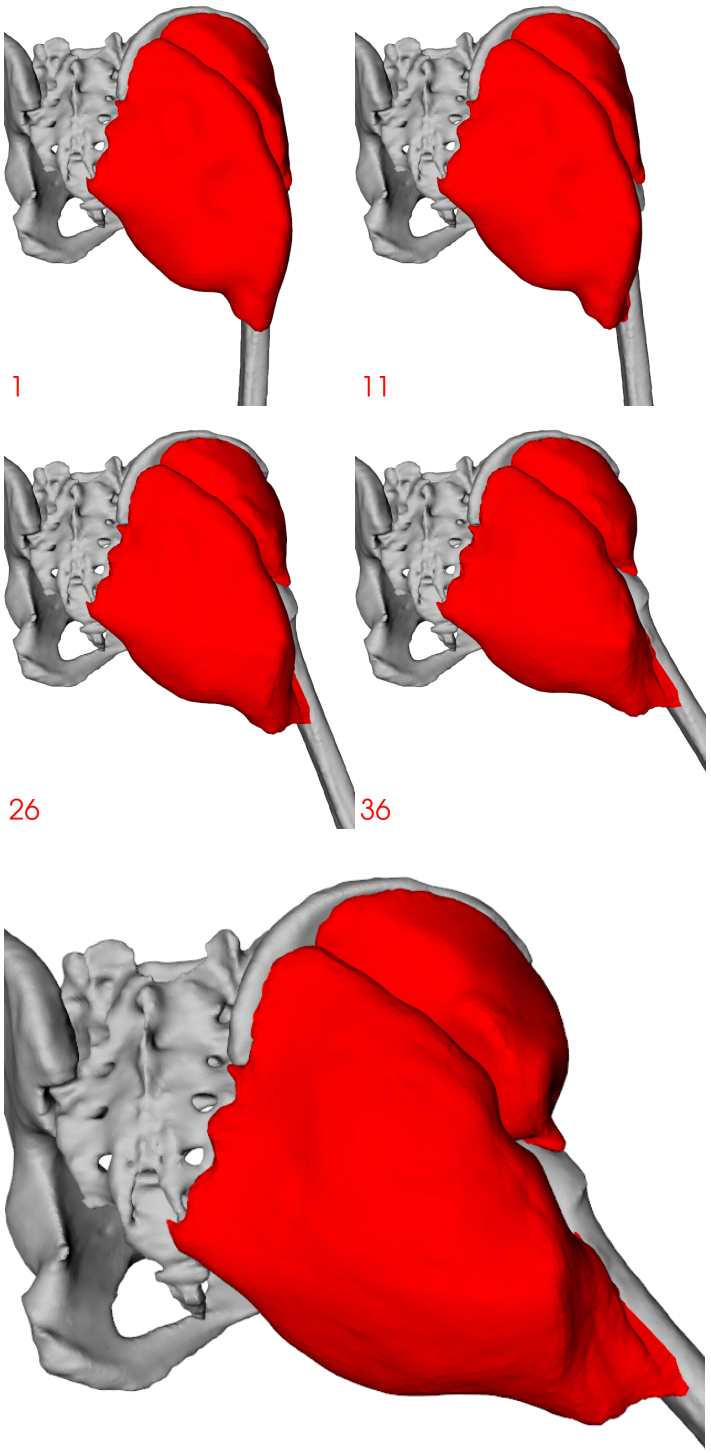


Figure A.12: Dorsal view of hip adduction

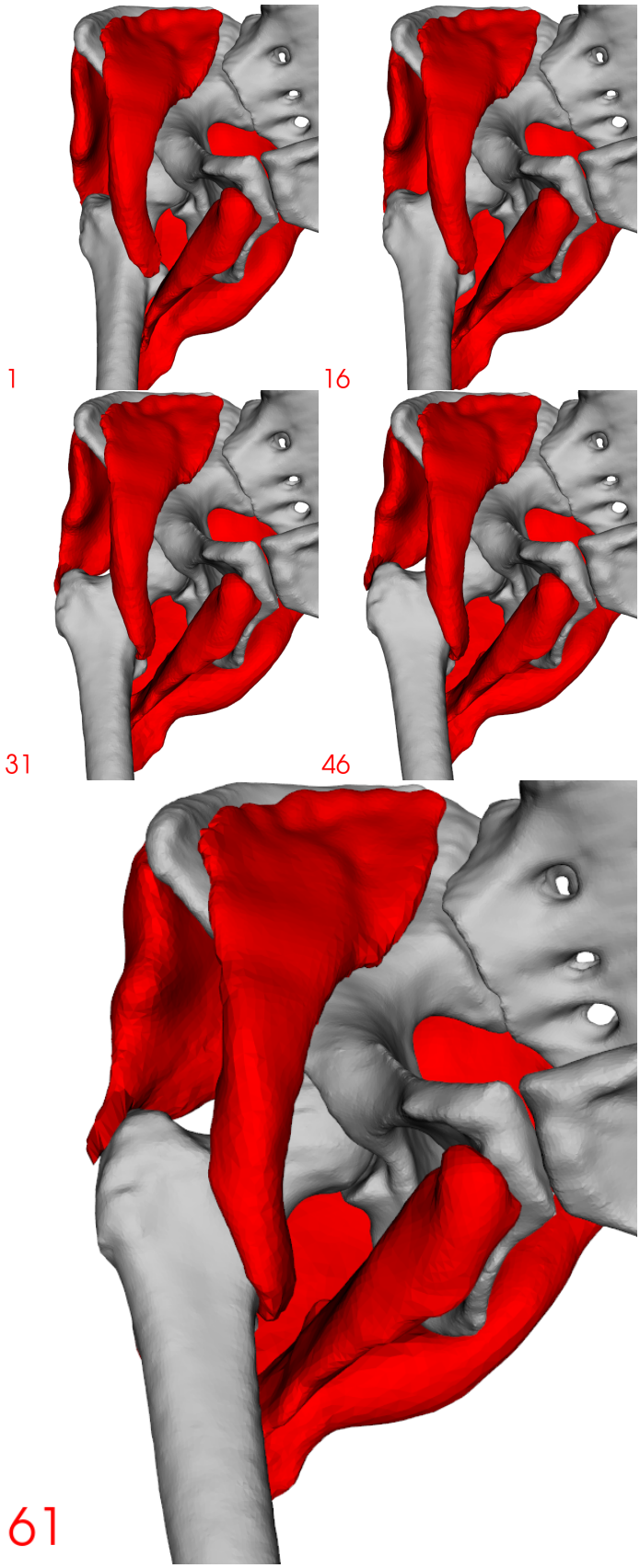


Figure A.13: Ventral-medial view of internal rotation

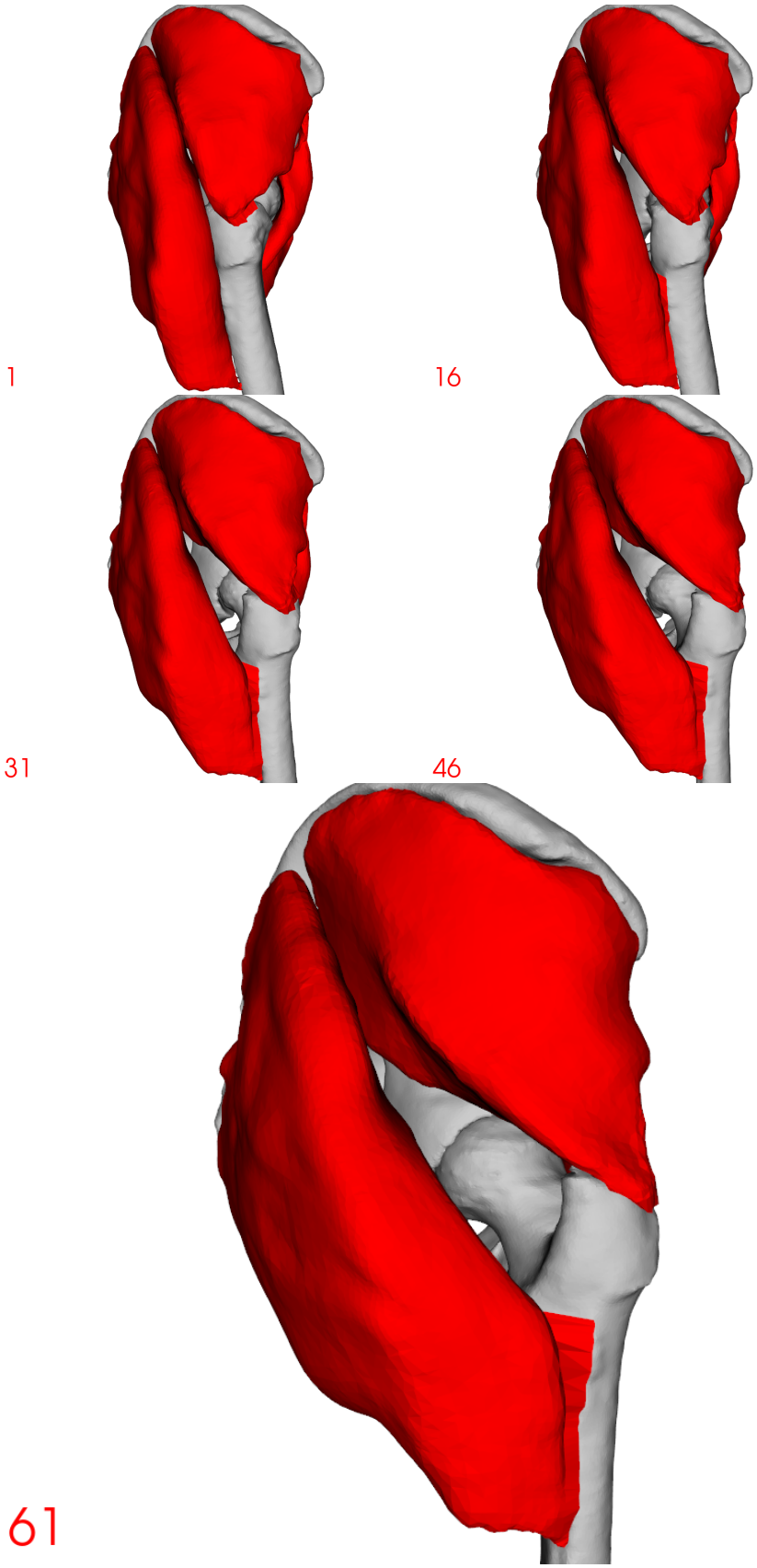


Figure A.14: Lateral view of internal rotation

# Tables B

Table B.1: Parameters used to obtain the verification results

| Parameter | Value |
|---|---|
| XPBD.h:**m_time_diff** | 0.001 |
| XPBD.h:**m_damp** | 0.9 |
| XPBD.h:**m_solver_iterations** | 70 |
| XPBD.h:**m_gravity** | 0 |
| XPBD.h:**m_virtual_edges_distance** | 0.02 |
| XPBD.h:**m_virtual_edge_update_period** | 5 |
| XPBDjoint.h:**s_K_neighbours** | 10 |
| XPBDjoint.h:**s_minimal_sarcomere_shortening_percent** | 0.7 |
| muscle_cd_cr.h:**m_resolution_factor** | 1 |
| muscle_cd_cr.h:**m_default_collision_threshold** | 0.002 |
| muscle_cd_cr.h:**m_zero_iteration_collision_threshold** | 0.002 |
| muscle_cd_cr.h:**m_default_max_iter** | 100 |
| constraint.h:**s_time_diff** | 1 |
| constraint.h:**s_beta** | 0.01 |
| constraint.h:**ms_do_not_solve** | 0 |
| constraint.h:**ms_distance_stiffness** | $1e^2$ |
| constraint.h:**ms_dihedral_angle_stiffness** | $1e^8$ |
| volumeConstraint.h:**m_pressure** | 1 |
| volumeConstraint.h:**m_time_diff** | 0.001 |
| volumeConstraint.h:**s_compliance** | 0 |
| SDF.h:**m_resolution** | (128,128,128) |
| SDF.h:**m_domain_margin_multiplier** | 1.5 |
| SDF.h:**m_collision_bone_margin** | 0.001 |

# User's guide C

In the folder `Input_data`, there is **one** important sub-folder and **three** important files the user should note.

- **MOT/** – the sub-folder contains motions files, which can be generated e.g. in a spreadsheet editor, and then inputted into the system.
- **setup_MuscleGeneratorTool.xml** – the main setup configuration file the results were done with.
- **setup_MuscleGeneratorTool_More_muscles.xml** – a setup file containing more of the muscles.
- **setup_MuscleGeneratorTool_fibres_low_res_low_count.xml** – a setup file with the four basic muscles with sparse fibres generated

A closer look at the **setup_MuscleGeneratorTool.xml** file reveals a few important elements that can be tuned.

- **motion_file** – specifies the used motion.
- **MuscleGenerator** – the user can add or remove muscles.
- **num_of_lines** – the number of inner fibres to generate.
- **line_res** – the resolution of each of the fibres.

These parameters can all be tuned to an extent.

In the **MOT** sub-folder, one can find all the verified motions plus a flexion up to 70 degrees and then back, while leaving time for stabilization. The user is invited to try that one as it gives interesting results.

The supplied electronic attachment also contains a folder called the `Application_and_libraries`, containing the script **RunInteraction.cmd**. By clicking this script, the user gets to choose from **10**, **30**, **50**, and **70** inner solver iterations the XPBD should have. Then, the application starts, where the user can navigate the

scene (using mouse manipulation, switching from **S**olid to **W**ireframe) and skip to the next frame using an *ESC* button. If the *ESC* key is pressed and then pulled up, the simulation stops at approximately the simulation the key is upped. If the key, is instead pressed repeatedly, the simulation steps queue and the number of clicks will be simulated, unless the calling process is stopped.

If the desire to change the **XML** setup file occurs, the user must open the script in a text editor, and change the first string in the file to that desired path.

To build the application, it is best to pull it from **GIT** gitlab.com/besoft/muscle-wrapping-2.0 (the branch containing Havlicek2024 in its name) and follow the directions there. But in other cases, the application can be built using **CMake** from the root **CMake** file on the path `Aplication_and_libraries\Sources\CMakeLists.txt`, as specified on the GIT page.

# Bibliography


[AP01] ANDERSON, Frank C.; PANDY, Marcus G. Static and dynamic optimization solutions for gait are practically equivalent. *Journal of Biomechanics*. 2001, vol. 34, no. 2, pp. 153–161. ISSN 0021-9290. Available from DOI: https://doi.org/10.1016/S0021-9290(00)00155-X.

[BMM17] BENDER, Jan; MÜLLER, Matthias; MACKLIN, Miles. A survey on position based dynamics, 2017. In: *Proceedings of the European Association for Computer Graphics: Tutorials*. Lyon, France: Eurographics Association, 2017. EG '17. Available from DOI: 10.2312/egt.20171034.

[Bet+13] BETTS, J. Gordon et al. *Anatomy and Physiology, Chapters: 11-1 Interactions of Skeletal Muscles, Their Fascicle Arrangement, and Their Lever Systems*. Houston, Texas: OpenStax, 2013. Available also from: https://openstax.org/books/anatomy-and-physiology/pages/1-introduction. Accessed: 2024 May, Access for free at https://openstax.org/books/anatomy-and-physiology/pages/1-introduction.

[BC13] BIZZI, Emilio; CHEUNG, Vincent C K. The neural origin of muscle synergies. *Front Comput Neurosci*. 2013, vol. 7, p. 51.

[Čer+23] ČERVENKA, Martin; HAVLÍČEK, Ondřej; KOHOUT, Josef; VÁŠA, Libor. Computerised Muscle Modelling and Simulation for Interactive Applications. In: *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - GRAPP*. SciTePress, INSTICC, 2023, pp. 214–221. ISBN 978-989-758-634-7. ISSN 2184-4321. Available from DOI: 10.5220/0011688000003417.

[DT18] DAO, Tien Tuan; THO, Marie-Christine Ho Ba. A Systematic Review of Continuum Modeling of Skeletal Muscles: Current Trends, Limitations, and Recommendations. *Applied Bionics and Biomechanics*. 2018, vol. 2018, p. 7631818. ISSN 1176-2322. Available from DOI: 10.1155/2018/7631818.

[De +18] DE PIERI, Enrico et al. Refining muscle geometry and wrapping in the TLEM 2 model for improved hip contact force prediction. *PLOS ONE*. 2018, vol. 13, no. 9, pp. 1–19. Available from DOI: 10.1371/journal.pone.0204109.

[Dom05] DOMINÉ, Sébastien. *Mesh Skinning*. Nvidia Corporation, 2005. Available also from: https://developer.download.nvidia.com/assets/gamedev/docs/skinning.pdf. Accessed: 2024 May.

[Gas+24] GASH, Matthew C.; KANDLE, Patricia F.; MURRAY, Ian V.; VARACALLO, Matthew. Physiology, Muscle Contraction. In: *StatPearls* [https://www.ncbi.nlm.nih.gov/books/NBK537140/]. Updated 2023 Apr 1. Treasure Island (FL): StatPearls Publishing, 2024. Accessed: 2024 Mar.

[Gui09] GUIBAS, Leonidas. *CS164: Introduction to 3D Rendering*. Stanford University, Department of Computer Science, 2009. Available also from: graphics.stanford.edu/courses/cs164-09-spring/Protected/09cs164-05-18_slides.pdf. Accessed: 2024 May.

[Hei+23] HEINRICH, Dieter; BOGERT, Antonie J. van den; MÖSSNER, Martin; NACHBAUER, Werner. Model-based estimation of muscle and ACL forces during turning maneuvers in alpine skiing. *Scientific Reports*. 2023, vol. 13, no. 1, p. 9026. ISSN 2045-2322. Available from DOI: 10.1038/s41598-023-35775-4.

[Hua+06] HUANG, Jin et al. Subspace Gradient Domain Mesh Deformation. *ACM Trans. Graph.* 2006, vol. 25, pp. 1126–1134. Available from DOI: 10.1145/1141911.1142003.

[Jan12] JANÁK, Tomáš. *Fast soft-body models for musculoskeletal modelling*. Tech. rep., University of West Bohemia, Faculty of Applied Sciences, 2012.

[KOA19] KAPANDJI, Adalbert; OWERKO, Carrie; ANDERSON, Alexandra. *The Physiology of the Joints - Volume 2*. .7th ed. Pencaitland, Scotland: Handspring Publishing, 2019. ISBN 978-1-912085-60-6.

[KČ21] KOHOUT, Josef; ČERVENKA, Martin. Muscle Deformation Using Position Based Dynamics. In: 2021, pp. 486–509. ISBN 978-3-030-72378-1. Available from DOI: 10.1007/978-3-030-72379-8_24.

[KK14] KOHOUT, Josef; KUKAČKA, Martin. Real-Time Modelling of Fibrous Muscle. *Computer Graphics Forum*. 2014, vol. 33, no. 8, pp. 1–15. Available from DOI: https://doi.org/10.1111/cgf.12354.

[Lob+22] LOBOVSKÝ, Libor et al. Experimental and Computational Study on Mechanical Stabilisation of Sacral Bone Injuries. In: *38th Danubia-Adria Symposium on Advances in Experimental Mechanics, DAS 2022*. Poros, Greece: Greek Society of Experimental Mechanics of Materials, 2022, pp. 1–2. ISBN 978-618-86278-0-2. Available also from: gsemm.gr/38_DAS/Extended_abstracts_of_the_papers_presented_at_DAS38_ISBN_978-618-86278-0-2.zip.

[MMC16] MACKLIN, Miles; MÜLLER, Matthias; CHENTANEZ, Nuttapong. XPBD: Position-Based Simulation of Compliant Constrained Dynamics. In: *Proceedings of the 9th International Conference on Motion in Games*. Burlingame, California: Association for Computing Machinery, 2016, pp. 49–54. MIG '16. ISBN 9781450345927. Available from DOI: 10.1145/2994258.2994272.

[Mül+07] MÜLLER, Matthias; HEIDELBERGER, Bruno; HENNIX, Marcus; RATCLIFF, John. Position based dynamics. *Journal of Visual Communication and Image Representation*. 2007, vol. 18, no. 2, pp. 109–118. ISSN 1047-3203. Available from DOI: https://doi.org/10.1016/j.jvcir.2007.01.005.

[NS13] NIETO, Jesus R.; SUSÍN, Antonio. Cage Based Deformations: A Survey. In: 2013. Available also from: https://api.semanticscholar.org/CorpusID:16001625.

[Óla17] ÓLAFSDÓTTIR, Jóna. *Muscle Responses in Dynamic Events - Volunteer experiments and numerical modelling for the advancement of human body models for vehicle safety assessment*. 2017. PhD thesis.

[OMB24] OMAR, Abdillahi; MARWAHA, Komal; BOLLU, Pradeep C. Physiology, Neuromuscular Junction. In: *StatPearls* [https://www.ncbi.nlm.nih.gov/books/NBK470413/]. Updated 2023 May 1. Treasure Island (FL): StatPearls Publishing, 2024. Accessed: 2024 Mar.

[Raj+16] RAJAGOPAL, Apoorva et al. Full-Body Musculoskeletal Model for Muscle-Driven Simulation of Human Gait. *IEEE Trans Biomed Eng*. 2016, vol. 63, no. 10, pp. 2068–2079.

[RMT16] ROKYTA, Richard; MAREŠOVÁ, Dana; TURKOVÁ, Zuzana. *Somatologie*. .7th ed. Praha: Wolters Kluwer ČR, 2016. ISBN 978-80-7552-306-8.

[Roș+21] ROȘCA, Andra Cătălina; BACIU, Cosmin Constantin; BURTĂVERDE, Vlad; MATEIZER, Alexandru. Psychological Consequences in Patients With Amputation of a Limb. An Interpretative-Phenomenological Analysis. *Frontiers in Psychology*. 2021, vol. 12. ISSN 1664-1078. Available from DOI: 10.3389/fpsyg.2021.537493.

[Soro5] SORKINE, Olga. Laplacian Mesh Processing. In: CHRYSANTHOU, Yiorgos; MAGNOR, Marcus (eds.). *Eurographics 2005 - State of the Art Reports*. The Eurographics Association, 2005. Available from DOI: 10.2312/egst.20051044.

[Tes+05] TESCHNER, Matthias et al. Collision Detection for Deformable Objects. *Computer Graphics Forum*. 2005. Available also from: https://inria.hal.science/inria-00394479.

[UD21] UCHIDA, Thomas K.; DELP, Scott L. *Biomechanics of Movement: The Science of Sports, Robotics, and Rehabilitation*. The MIT Press, 2021. ISBN 9780262044202. Available also from: https://mitpress.mit.edu/9780262044202/. Accessed: 2024 May. Used lecture videos available at youtube.com/watch?v=VbUNOFgYcKI&list=PL_uk_kfAmFLrtzEfv6njXooOPae3jI1q6.

[Uhl+22] UHLRICH, Scott D.; JACKSON, Rachel W.; SETH, Ajay; KOLESAR, Julie A.; DELP, Scott L. Muscle coordination retraining inspired by musculoskeletal simulations reduces knee contact force. *Scientific Reports*. 2022, vol. 12, no. 1, p. 9842. ISSN 2045-2322. Available from DOI: 10.1038/s41598-022-13386-9.

[Wad+14] WADE, Sally; STRADER, Cynthia; FITZPATRICK, Lorraine; ANTHONY, Mary; O'MALLEY, C. Estimating prevalence of osteoporosis: Examples from industrialized countries. *Archives of osteoporosis*. 2014, vol. 9, p. 182. Available from DOI: 10.1007/s11657-014-0182-3.

# Overview of abbreviations

1. PBD – Position Based Dynamics, a method for simulating deformable and rigid bodies defining constraints based on immediate vertex positions.
2. CNS – Central Nervous System, brain and spinal cord.
3. ATP – Adenosine TriPhosphate, a nucleotide that provides energy to living cell processes.
4. ACh – AcetylCholine, a neurotransmitter.
5. SNARE – Soluble N-Ethylmaleimid-sensitive fusion protein Attachment protein REcepetor, a protein making the ACh vesicles fuse within cell membrane on a neuromuscular junction.
6. AP – Action Potential, electrical potential as a result of cell membrane depolarisation.
7. DHP – DiHydro Pyrine, a voltage-gated calcium channel.
8. RyR – Ryanondine Receptor, a receptor within the sarcoplasmic reticulum.
9. ROM – Range Of Motion, the limits within bone rotation in the respective degree of freedom.
10. ° – a degree, unit of a rotation.
11. EMG – ElectroMyoGraphy, a measuring technique to detect electrical responses to stimuli in muscles or on their surface.
12. LiDAR – Light Detection and Ranging, a remote sensing method to collect depth-information point clouds.
13. 3D – Three-dimensional, having the dimension of three.
14. LE – Laplacian Editing, a deformation method based on the Laplacian operator.

15. `MS` – Mesh Skinning, a deformation method based on the internal skeleton.
16. `CBD` – Cage-Based Deformations, a deformation method based on the outer skeleton.
17. `FEM` – Finite Element Method, a deformation method based on discretizing the object.
18. `MSS` – Mass Spring System, a deformation method based on a set of elastic strings.
19. `XPBD` – eXtended PBD, a version of PBD where the constraints are not dependent on the simulation time.
20. `BSP` – Binary Space Partitioning, a binary tree used for recursive space subdivision to accelerate collision detection.
21. `BVH` – Bounding Volume Hierarchy, usually a tree structure used to accelerate collision detection.
22. `AABB` – Axis-Aligned-Bounding-Box, a rectangular prism parallel to one of the system axes in each of its faces, used for collision detection acceleration.
23. `SDF` – Signed Distance Field, information about distances to a watertight surface, where the information about outside and inside is known at every sampled position, used for collision detection acceleration.
24. `CRTP` – Curiously Recurring Template Pattern, a design pattern that allows static polymorphism in the C++ programming language.
25. `GPU` – Graphics Processing Unit, a processor originally specialised for graphics rendering.
26. `PSTL` – Parallel Standard C++ Library, a C++ programming language standard library parallelization library.
27. `SLO` – Smallest-Last Ordering, a strategy to accelerate greedy graph colouring by presorting the nodes by vertex degrees in a descendent fashion.
28. `CDP` – Contact Distance Proximity, the distance of a virtual edge where it can be deemed to approach collision.
29. `JCU` – Joint Control Unit, the abstraction over a joint and a bone inside of it capable of rotation relative to the joint, used a structure.

30. `SOT` – Static Optimization Tool, a tool provided by the OpenSim software to estimate muscle activations during movement.
31. `PCSA` – Physiological Cross-Sectional Area, the cross-sectional area of a muscle fibre, usually used in biomechanics.
32. `kd-tree` – K-dimensional tree, a structure used for K-nearest neighbours search acceleration.
33. `RBF` – Radial Basis Functions, and interpolation technique based on non-linear distance.
34. `SIMD` – Single Instruction Multiple Data, a set of instructions for a specific software architecture utilising vectorisation computation acceleration.
35. `CPU` – Central Processing Unit, the main processor on a computer.
36. `XML` – Extensible Markup Language, a structured data format using a hierarchy.

# List of Figures

# List of Tables

# List of Listings