University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master Thesis

Soft-body Deformation in Musculoskeletal Modelling

Pilsen, 2012 Tomáš Janák

I hereby declare that this diploma thesis is completely my own work and that I used only the cited sources.

Pilsen…… ……

Tomáš Janák

Acknowledgments

I would like to express my gratitude to Doc. Ing. Josef Kohout Ph.D. for offering me the opportunity to participate in the VPHOP project, for his supervision of my work and for all the advices, explanations and suggestions he gave me. A big thank you belongs to my VPHOP project colleague Yubo Tao Ph.D. for implementing part of the designed solution and the commitment he has shown during our cooperation.

A special thank you goes to my parents for the endless support they are providing me.

Abstract

Soft-body models represent elastic deformable objects in a virtual environment, which makes them particularly appropriate for a simulation of objects made of organic materials. Simulation environments created for medical purposes are the most usual applications that employ soft bodies. This thesis presents a model of muscles for one of such environments. The model is based on the mass-spring systems, which use point-mass particles connected by fictional springs to represent the deformable object. In this particular case, the particles are obtained by sampling the muscle fibres that stretch through the interior of the muscle and then connected by springs using a given pattern. Several spring layout patterns, as well as various different parameters of the mass-spring system, were tested in order to find out the ones most suitable for the muscle models. Further on, a mechanism for collision detection and response suitable for the purpose was designed, implemented and tested. The mechanism is able to handle collisions between a rigid and a soft body (a bone and a muscle) as well as between two soft bodies (two muscles). The model aims for interactivity rather than perfect physical accuracy of the model. The solution is a part of the EC funded project VPHOP - The Osteoporotic Virtual Physiological Human (FP7-ICT-223865) that is dedicated to improvement of the effectiveness of osteoporosis prediction and treatment.

Table of Contents

1. Introduction

When modelling stiff objects, the models can be divided into two groups – rigid and non-rigid (soft) bodies. The soft-body simulations are generally more complicated than simulation of rigid bodies, as some deformations of the objects may, and do, occur. They have been studied in the field of computer graphics for many years as they are required in numerous fields, such as animation in entertainment (video games, movies), cloth simulation, surgical training simulation or many medical purposes in general. One of these medical purposes is also the topic of this thesis.

The algorithms and program equipment that will be described in this thesis were developed for the VPHOP – The Osteoporotic Virtual Physiological Human (FP7-ICT-223865) project. The project's aim is a development of tools and methods that could be used for early recognition of osteoporosis' symptoms and subsequent prevention or treatment. Osteoporosis is a disease that significantly decreases the density of bone tissue, which results in higher liability to fractures. The VPHOP is funded by the European Commission and for a good reason – according to the International Osteoporosis Foundation [9], one in three women and one in five men are at risk of an osteoporotic fracture. There are approximately four million osteoporotic bone fractures happening every year [26] in the states of European Union. Moreover, there is 86% chance of having another fracture after the first one. The estimated cost for treatment of these injuries is 30 billion euro per year. According to current predictions, these numbers will double by the year 2050.

The main problem in diagnosis (and therefore prevention) of osteoporosis is the variety of patients. The probability that the osteoporosis will make itself felt is dependant on numerous factors such as constituency of the bone tissue, which influences the strength of the bone, the musculoskeletal anatomy of the patient, which influences daily load affecting the bones, and many others. Therefore, the main goal of the project is to create a model that could be parameterized for each patient in order to allow incorporation of all the significant characteristics of the patient and correct deduction of their impact.

There are currently (as of spring 2012) twenty two partners involved in the VPHOP project, from both academic and industry grounds. University of West Bohemia is one of them, participating in the design and implementation of software equipment. The goal is to create a framework that would allow a trained specialist to feed data from conventional diagnostic imaging methods, such as MRI or CT, to the program. Based on this data, the program should create the personalized musculoskeletal model of the patient. Finally, using the created model, the framework would enable to process simulations of various scenarios that would test the strength of the patient's bones and their liability to osteoporotic fractures under various workloads. Suitable treatments could then be applied to the patient based on the results of these simulations.

That is obviously a large scale project and only a rather small part of it is solved in this thesis. The aim is to create a credible model of muscles, which are naturally a part of the musculoskeletal model and their behaviour has an impact on the whole simulation. The model must be deformable, as muscles are elastic. It must allow correct interaction with the surrounding bones and other muscles as well. Another demand is speed. Although full interactivity is not needed, as the specialist will be concerned mainly with the result of the simulation, the simulation should be processed in times close to real time. In terms of

Computer Graphics, a soft body model that is capable of detecting collisions between soft (muscle vs. muscle) and soft and rigid bodies (muscle vs. bone) in real time is sought.

Chapters 2 and 3 focus on methods applicable for modelling of soft bodies and collision handling mechanisms applicable on these models, respectively. Chapter 4 contains the program model designed for the solution and various tests follow in chapter 5 and their results are discussed there as well. The conclusion and final thoughts are presented in the last chapter. Material related to the actual implementation can be found in the Appendix.

2. Soft body models

In context of computer graphics, a soft body represents a deformable object, i.e. an object that changes its shape as external forces apply pressure on it. However, a soft body retains its original shape to a certain degree, depending on the magnitude of the applied forces – it "resists" those forces and, unlike fluid, it tries to maintain its original shape as much as is possible. The soft body is almost always defined in its original shape, usually called "rest shape/position". Soft body models are well fitted, and mostly used, for representation of any soft organic materials (muscles, fat, body organs, vegetation etc.) as well as clothing and fabric.

Soft body dynamics is a discipline that focuses on realistic simulations of soft bodies. As one or multiple simulated soft bodies interact with their virtual environment, the soft body model is responsible for shaping the objects the way they would in real world. The most important implication of this is that a soft body model should be able to represent physical properties of the material it is made of. However, ensuring physical accuracy of such simulation may be very demanding in terms of computational power.

Consequently, various approaches to modelling of deformable objects differ in tradeoffs between physical accuracy and interactivity. According to [7], they can be divided into two basic groups – geometric modelling and physically based modelling. A more recent state of the art survey [13] offers a different point of view, branding the methods as "heuristic approaches", "continuum mechanical approaches" and "hybrid approaches". The heuristic approaches operate on the assumption that methods, which try to accurately simulate the material of the object (e.g. Finite Element Method), are too complex for interactive applications. Therefore, heuristic methods do not try to model the actual structure of the simulated object, but employ some ad-hoc modelling approach that will produce acceptable visualization of the soft body while being able to run the simulation interactively. Examples of such methods, which will be further discussed below, are geometric models and mass-spring systems. Needles to say, continuum mechanical approaches stands on the opposite side and hybrid models try to combine the advantages of both approaches and *"are characterized by dividing a deformable object into different sections according to the expected kind of interaction with each of these sections, and to model each one of these with an appropriate model"* (direct excerpt from [13]).

It is interesting to note, that [7] talks about mass-spring model as a physically based model, while [13] groups it with the ad-hoc heuristic models. While it is true that the mass-spring system model is based on certain physical principles, there is not any direct connection between the model and the real world object it is trying to simulate; i.e. the parameters of the mass-spring system (like the stiffness or damping of springs, see 2.3.) are difficult to derive from the material properties and usually are chosen experimentally. In any way, as mass-spring system is the approach that was chosen for the actual realization of the simulation framework created for this thesis, it is described more thoroughly than other methods in its own subchapter 2.3. The other subchapters cover the topic of aforementioned geometric models (2.1.) and various physically based models (2.2).

Before describing various soft body models, let us first specify the objects that will be modelled in this thesis. There are actually two models of the muscle employed in the VPHOP project. One is a closed surface, represented by a triangular mesh, and the other is a volumetric model of the muscle fibres as displayed in figure 2.1. The fibre model is

actually more important for the diagnosis than the surface, therefore a correct deformation of the fibres is important. However, the base on which the fibres are generated is the surface model. This implies two possible approaches to the soft body model.

One is to use the triangular surface as the basis for the soft body model, using either only the surface itself (a boundary model – see sections 2.2 and 2.3) or sampling its interior to create a volumetric model (e.g. create a tetrahedral mesh from the triangular). After deforming it, the fibres could be generated from the result. The other approach is to use the fibres as the basis. This way, the fibres will be deformed directly, without the need of calling the procedure that generates them in every step of the simulation, therefore this approach was chosen. As such, the data used to build the soft body model will be volumetric. For this reason, main emphasis in the following sections will be on models based on volumetric data rather than boundary models.

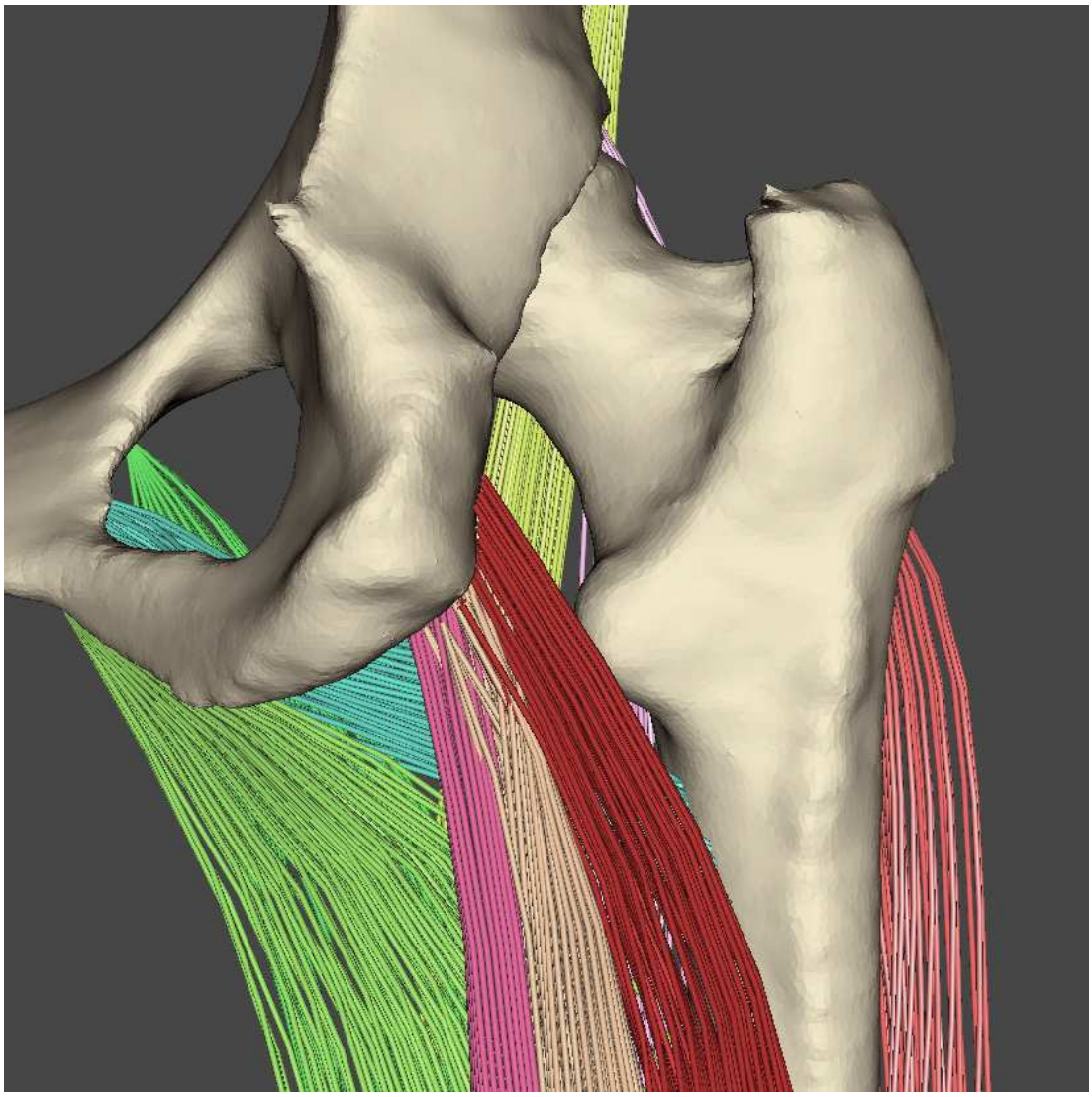

Figure 2.1: Muscle fibres of several muscles (colour coded) attached to pelvis and thigh.

2.1. Geometric models

When talking about soft bodies, the most attention is usually drawn to the need for correct depiction of the behaviour of the material the modelled object is made of. That is why it is

quite intuitive to think about soft bodies as about volumetric models. However, for visualization and haptic purposes, only the surface of the object is needed. Moreover, surface models obviously have lesser computational requirements than volumetric models. Those facts together with a relatively long history of various surface modelling techniques used in computer aided geometric design (CAGD) make it unsurprising that the initial attempts to model soft bodies were using the tools of CAGD.

Bézier surfaces started a new era of CAGD in the early 1960s and other more or less similar surfaces began to be used in CAGD after that, such as general B-splines and NURBS. Such surfaces are defined by a set of "control points". By moving these control points, the shape of the surface changes. In case of readers' interest, textbooks like [1] will provide more detailed information on modelling with spline surfaces. Other possibility to create a deformable geometrical object is called free-form deformation [18]. This approach can be applied to spline surface as well as to traditional polygonal models, parametric patches or implicit surfaces. The reason for such universality is that instead of modifying the object itself, the object is enclosed in a simple (e.g. cubical) grid of control points and these points are modified. This warps the space inside the grid and therefore the enclosed object as well (for details please refer to [18]).

These approaches allow a skilled designer to quickly modify the object without having to remodel the parts that he wants to modify. After having modelled the object in its rest position (original, undeformed shape), the designer can easily deform it using the aforementioned tools, thereby simulating influences of external forces. However, it is clear that such approach can hardly be called a soft body *simulation*, as the resulting shape of the object is dependant solely on the input of the designer. Even if the designer was able to immaculately capture the behaviour of the model's material when undergoing deformation, this approach would clearly be very costly and definitely could not be done in real time. For some purposes, like animation for movies, it is possible to have a designer (animator) model all the deformations manually. But in most cases of soft-body deployment it is needed to be able to process the deformations automatically.

2.2. Physically based models

During late 1980s, Terzopoulos et al. introduced "active" spline models ([20], [21]), which can be considered as a bridge between geometric and physically based models of soft bodies. While creating an innovative approach to soft body simulation, their work actually originates from the fundamental principles of splines. They pointed out that splines are based on elasticity theory and it is for instance possible to describe their strain energy or other characteristics using the elasticity theory and differential equations.

The method by Terzopoulos et al. formulates a potential energy over the simulated body (or surface or curve in 2-D or 1-D cases) and then uses Newtonian motion equation (2.1) to simulate the dynamics of the body in time. The equation (2.1) is an ordinary second order differential equation differentiated in time. It is describing the motion of a single point $a = [a_x, a_y, a_z]$ on the simulated body, which in a given time t occupies position $s = [s_x, s_y, s_z]$ (therefore $a = s_{t=0}$). The symbol $\ddot{s}$ stands for a second time derivative and $\dot{s}$ stands for first time derivative; μ is the mass density of point a, γ is the damping density of point a. The remaining term on the left hand side of the equation is the variational derivative of the potential energy of deformation $\varepsilon(s)$ (also called "elastic energy"), which will be discussed further. F is the sum of external forces applied to point a. By solving this equation for all

points of the object over time, the movement of those points, and therefore also the deformation of the object, is obtained.

$$\mu\ddot{s} + \gamma\dot{s} + \frac{\delta\varepsilon(s)}{\delta s} = F \tag{2.1}$$

The elastic energy defines the internal elastic forces (please refer to textbooks such as [11] for details). Terzopoulos et al. measured the deformation using concepts from the differential geometry of curves, surfaces and solids. The shape of three dimensional bodies can be described solely by the change of distances between nearby points. The distance *D* of such two points a_0 and (a_0 + ***d****a)* can be described by equation (2.2) (using the same notation in equation (2.1)).

$$\begin{aligned} D &= (da)^T \cdot G(a_0) \cdot (da) \\ where \quad G_{ij}(a_0) &= \frac{\partial r}{\partial a_i}(a_0) \cdot \frac{\partial r}{\partial a_j}(a_0) \end{aligned} \tag{2.2}$$

The matrix *G* is called the “first fundamental form” of “metric tensor”. If two three dimensional bodies have the same metric tensor for each point *a*, they must have the same shape (they can however differ by some rigid motion, e.g. simple translation). Note that in case of objects with fewer dimensions, curvature and torsion tensors have to be taken into account as well. Finally, the elastic energy can be defined using equation (2.3) (integrating over all points *a* of the object):

$$\varepsilon(s) = \int \left\| G - G^0 \right\|_\alpha^2 da_x da_y da_z \tag{2.3}$$

The G^0 is the metric tensor in the rest pose of the object; the $\|\ldots\|_\alpha$ is a weighted matrix norm. In the actual implementation, the integral in equation (2.3) changes to a finite sum over all points of the mesh. The metric tensors are computed for each point. Using them, the elastic energy is computed afterwards and then new positions for the points are obtained by numerically integrating the system of equations (2.1) through time (some simplification occur, e.g. when computing the variational derivative of the elastic energy, refer to [21] for details).

This approach still incorporated some parameters, which changed the way the material behaves, that had to be chosen experimentally, therefore there was not a guaranteed high level of physical accuracy. As the aim was mainly computer animation, it was not actually a priority at the time. However, the computational cost of this method is rather high as well, so although it was one of the pioneering approaches in soft body simulation, more suitable solutions can be found today.

If the aim is mainly physical accuracy, approaches based on continuum mechanics are one of these more suitable solutions. Continuum mechanics describes the volume of an object using a set of partial differential equations (PDE), each equation belonging to one volume element of the object after it has been discretized. The equation describing elastic materials

is called the Navier's equation (2.4), where ∇ stands for gradient, $\nabla\bullet$ stands for divergence, λ and μ are the Lamé constants, which describe material properties, u is the displacement vector of a given point in respect to the rest position (i.e. $u = s_t - s_0$) and F are the net external forces applied to the point. The unknown is the displacement of the point.

$$(\lambda+\mu)\nabla(\nabla\cdot u)+\mu\nabla^2 u = F \tag{2.4}$$

Probably the most popular mathematical tool to solve such sets of PDEs is the finite element method (FEM). It is used to transform a PDE into a set of ordinary differential equations, which are afterwards solved using some numerical scheme. The entire domain Ω (in this case the deformable object) is, usually irregularly, divided into a set of elements. The spatially continuous function (2.4) is approximated in each element by a polynomial function. Equation (2.5) formulates this approximation formally. $\tilde{u}(\Omega,t)$ approximates the continuous displacement function by summing basis functions b_i defined for the set of i elements[17].

$$\tilde{u}(\Omega,t) = \sum_i u_i(t) b_i(\Omega) \tag{2.5}$$

After substituting the approximation (2.5) into equation (2.4), it is numerically solved. The resulting equation system for a dynamic object, again based on the Newtonian motion equation, then takes the form of equation (2.6), in which M is the mass matrix (diagonal matrix with masses of the elements), D and K are the damping and stiffness matrices respectively, which capture the material properties, u are the displacements and F is the force matrix.

$$M\ddot{u} + D\dot{u} + Ku = F \tag{2.6}$$

During the simulation, most of the displacements u_i are unknown and no direct force acts upon them (therefore $F_i=[0; 0; 0]$). For instance, after a deformable body collides with a rigid body, only displacements for the contact elements are known. Therefore, it should be obvious that in a general case, the equation system cannot be solved exactly. Instead a solution that minimizes the residue, which the approximation creates when substituted into the original PDE, is sought.

The computation time is prescribed by the algorithm used to solve the equation system, therefore various solutions based on FEM can achieve various speed. Moreover, the precision of the approximation can be different (see [17] for details) – simple, linear approximation functions are faster, but fail to maintain realistic shape of the object when undergoing large deformations. Nevertheless, generally all of basic FEM approaches are deemed for off-line usage, as none can achieve real-time speed for objects detailed enough to be useful in real applications. It comes as no surprise then that there can be found many proposals for simplifications of the FEM for soft body simulation.

A very popular speed-up technique in FEM simulations is using the so called "explicit" FEM. As [17] explains, in the explicit FEM both the masses and the internal and external forces are lumped to the vertices of the object. The elements act similarly as springs in a

mass-spring system, connecting all adjacent mass points. The motion of each vertex is then computed locally, based only on the surrounding vertices and elements. This way, instead of solving the large system (2.6), each element is solved independently using this local approximation, which results in faster computational time. However, this approximation also introduces some errors and so in order to achieve pleasing results, the mesh has to be discretized into more elements, which on the other hand slows the computation.

To fight this drawback, Debunne et al. [5] proposed a method which employs level-of-detail concepts, well known from other branches of computer graphics. The basic idea is to have more elements in the region of the object, where the deformation is large, e.g. in contact regions during collisions, and less in regions which are unlikely to be influenced by the source of the deformation. Various other research groups embraced this concept as well, for example [16], which also uses FEM with some enhancements or [8], where an adaptive mass-spring system is used.

One of the crucial problems of these methods is ensuring that the object stays consistent across various levels of detail and as Debunne et al. states, this makes mass-spring system not very convenient for the purpose, as there is no physical model to refer to in order to find what changes in parameters of the system will simulate the same behaviour at different resolutions. Therefore, explicit FEM is used in their approach and the level-of-detail techniques are implemented in such a way that by changing the cost/precision ratio a guaranteed frame rate is achieved. In [5] they state that this approach yields up speed up factor of 5 to 20 when compared to fixed resolution approaches. That, however, still limits the method for rather small objects when real-time computation is wanted. Moreover, although the method is nicely fitted for example for virtual surgery, where the tool truly makes only local deformations, it is not very fit for some other purposes, like the musculoskeletal model that is the subject of this thesis. In it, the deformed object – a muscle – will always be surrounded by other muscles and bones and therefore undergoing deformations (by colliding with those objects) more or less along its whole surface, dissipating the advantage of an adaptive model.

Another method based on FEM is called the boundary element method (BEM). As the name implies, this method reduces the problem to the boundary of the object, so unlike FEM it does not consider the volume of the object but only its surface. The first-hand implications are quite obvious – only homogenous objects can be simulated and fracture or tearing is more difficult to simulate. However, as the problem loses one dimension, speedup is achieved. A comparison between both methods (as well as detailed description of the BEM) is presented in [19]. The authors performed tests on simple mesh of a cube with varying number of vertices in orders of hundreds to thousands and displaced some of its vertices to simulate a deformation. When 10% of the vertices were displaced, the FEM system took several seconds to stabilize, while when 30% were displaced, it took several tens of seconds. The BEM method was generally ten times faster. However, this test does not takes into account the plausibility of the deformation, which would have to be tested on more complex objects, and it can be expected that the BEM could introduce some artefacts.

2.3. Mass-spring systems

Mass-spring system (MSS) is probably the simplest deformable model available. As the name implies, it consist of point masses connected by springs. So unlike FEM based methods, which are developed on a basis of some equations, the theory of MSS starts directly with a discrete model.

The point masses, or particles, of the MSS are defined by their mass and position. The springs are defined by stiffness and damping coefficients, the two particles it connects and the rest length, which is simply the length the spring has in the rest position of the model. When a particle is moved, the springs, which are connected to it, change their lengths. Every spring tends to return to its rest length though. This introduces forces acting on the end points of the spring. Hooke's law describes this force by a "spring equation" (2.7). F is the resulting force, k is the stiffness of the spring, l_{ij} is the length of the spring connecting i-th and j-th particle while the zero superscript again denotes the rest pose.

$$F_{ij} = k(|l_{ij}| - |l_{ij}^0|)\frac{l_{ij}}{|l_{ij}|} \tag{2.7}$$

The movement of the particles can be described by Newtonian mechanics as in other methods mentioned in section 2.2. When only one spring and one particle is accounted for, it takes the form of equation (2.8), where m is the mass of the observed particle, c is the damping coefficient of the spring, k is again the stiffness coefficient and x is the position of the particle, with appropriate time derivatives.

$$m\ddot{x} + c\dot{x} + kx = 0 \tag{2.8}$$

The solution of this ordinary differential equation is equation (2.9), C_1 and C_2 are the constants of integration and λ_1 and λ_2 are the roots of the characteristic equation (2.10).

$$x = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t} \tag{2.9}$$

$$m\lambda^2 + c\lambda + k = 0, \quad \lambda_{1,2} = \frac{-c \pm \sqrt{c^2 - 4km}}{2m} \tag{2.10}$$

As [29] explains, the value of the discriminant decides, whether the system will oscillate (negative discriminant – "under-damping") or not (positive – "over-damping"). If it is zero, the system will not oscillate and will stabilize in minimum possible time. This state is called the "critical damping" and obviously is very sought, as the fastest stabilization in fact means the fastest simulation of the soft body. It can be seen, that to achieve a zero discriminant, the damping coefficient c must be equal to $\sqrt{4km}$.

However, note that equation (2.8), (2.9) and (2.10) were all derived for a system of one particle connected to one spring, which is supposedly connected to an immovable object on its other end. In the case of real MSS, the situation is much more complicated. Zelený [29] devised and tested a formula for approximation of the damping coefficient for complex MSS. The formula is based on a simple principle of averaging the parameters. Therefore, instead of using stiffness coefficient k, an average stiffness k_a is computed and used instead. Also, as a spring always connects two mass points, the mass m in the equation should be doubled. His test proved the formula quite accurate in terms of minimizing the stabilisation time. Equation (2.11) is that formula, with k_a being the average stiffness. The

formula was devised for a system in which every particle is connected with six other. Apparently some assumptions about the order of the spring length (which is used for determining the stiffness, see below) were made. It can be seen that the formula reflects that by using some additive constant "2" and that unfortunately means that an application of this formula to a general MSS is not straightforward.

$$c = \sqrt{4k_6 2m}, \quad k_6 = 2 + \frac{4}{k_a} \tag{2.11}$$

This finding only underlines the statement made at the beginning of chapter 2 that the parameters of MSS are often difficult to derive and usually have to be found experimentally. In case of the stiffness parameter a generally good approach is to start with the inverse of the length of the spring (which can be then multiplied by some "material" constant). The reasoning for this is that this way the behaviour of the springs, and therefore the overall plausibility of the deformation, remains consistent through the whole object even for springs of different length.

As was stated before, equation (2.8) is valid only for one particle. Equation (2.12) shows the actual form that needs to be solved for every particle i in a general MSS, with F^e representing external forces acting on the particle, F_{ij} the force computed using equation (2.7) and N_i the set of particles, to which particle i is connected by a spring.

$$m_i \ddot{x}_i + c\dot{x}_i + \sum_{\forall j \in N_i} F_{ij} = F_i^e \tag{2.12}$$

To obtain an exact solution of the differential equation (2.12), it has to be integrated in time. Various integration schemes have been tested [22] and Verlet integration emerged as the most suitable for application in MSS. Moreover, it is quite simple to implement. It discretizes time by replacing the derivatives by differences between sufficiently small steps dt. The step dt is an additional parameter of the system which contributes heavily to its behaviour – a too small step will result in lengthier computations while too big steps will result in divergence of the integration scheme and therefore the system itself (i.e. it will not be able to achieve a stable position). Equation (2.13) depicts the substitution of differences and derivations.

$$\ddot{x}_i(t) = \frac{\frac{x_i(t+dt) - x_i(t)}{dt} - \frac{x_i(t) - x_i(t-dt)}{dt}}{dt} = \\ = \frac{x_i(t-dt) - 2x_i(t) + x_i(t+dt)}{dt^2} \tag{2.13}$$

$$\dot{x}_i(t) = \frac{x_i(t) - x_i(t-dt)}{dt}$$

Equation (2.14) is then the resulting formula for particle position in the next time step $t + dt$. The new position is computed solely from two previous positions, the current position t and previous position $t - dt$. That implies that in the actual implementation, only the positions have to be stored – the velocities and accelerations are integrated in the formula already.

$$x_i(t+dt) = \frac{F_i^T(t)}{m_i} dt^2 + 2x_i(t) - x_i(t-dt)$$
$$F_i^T(t) = F_i^e - \sum_{\forall j \in N_i} F_{ij} - c\frac{x_i(t) - x_i(t-dt)}{dt} \qquad (2.14)$$

The last defining quality of a MSS is the layout of the particles and springs itself. In FEM based approaches, the whole volume of the simulated object is divided usually by tetrahedralization and each tetrahedron is used as an element; or in case of BEM the volume is replaced by surface and tetrahedra by triangles. To sample particles and spring of a MSS a similar approach can be used, setting the vertices of the tetrahedralization / triangulation as the particles and the edges as springs.

In the context of musculoskeletal modelling however, the volume of a muscle is not described by tetrahedralization, but instead by muscle fibres. Therefore, the particles are obtained by sampling these fibres. There are numerous options on how to connect those particles by springs. The most straightforward approach is to build some tetrahedralization on the sampled particles and use its edges as springs. Another possibility, used in [29], is to handle the particles as if they were vertices of some template layout, e.g. regular cubical grid. The edges in the template then form the springs. For example, in a simple cubical grid template, each particle will be connected by a spring with its six neighbours. Or, instead of only the direct neighbours, it can be connected also with its “diagonal” neighbours to create more dense net of springs. Both possibilities are displayed in figure 2.2.

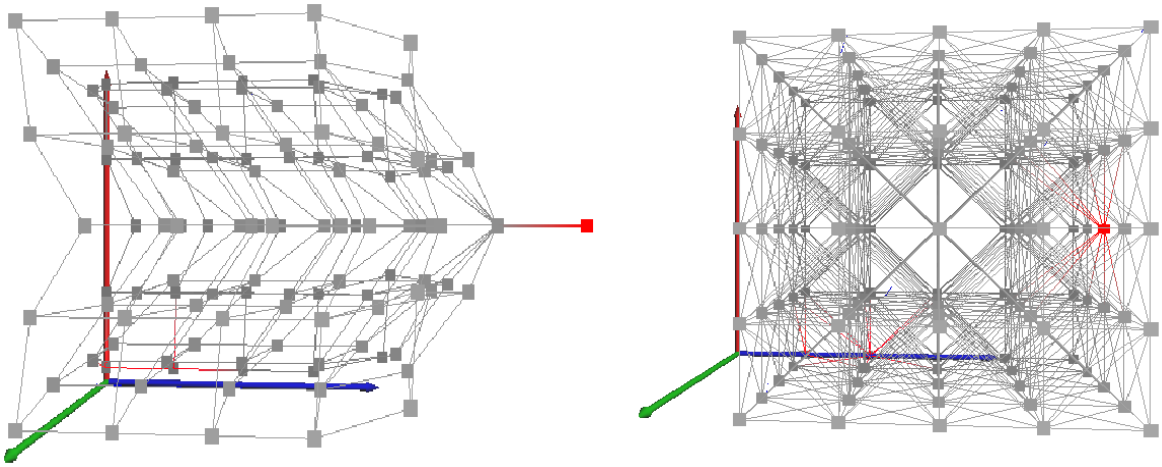

Figure 2.2: An example of mass-spring templates. On the left image, each mass point is connected with six neighbours (except for points on the border obviously), on the right image, each is connected with all 26 neighbours, i.e. springs are also on all diagonals of the cube elements. Image taken from [29].

Figure 2.2 also shows that the end “stable” states for different templates can differ. The red point on the figure was displaced and set as immovable. In the case of sparser spring configuration, the “stable” state does not guarantee preservation of the original shape (which was a simple cube). The reason is that although the distances between points connected by springs are preserved, that does not have to be true for points not connected by springs. Note however, that the same holds true when the “tetrahedralization” approach is used. The connectivity of the model (i.e. number of springs per particle) is therefore an important parameter. The higher it is, the better will the original shape be preserved, but the computational time and memory consumption will be higher as well.

It is also worth noting that as the particles are sampled along the fibres, the correspondence between the particles and the original surface mesh is lost. Therefore, it has to be somehow established artificially in order to be able to actually deform the surface mesh based on the movement of its particles, e.g. use the same displacement for the mesh vertices as is used for the closest particle(s) to that particular vertex (see 4.4 for details).

Obviously, the more springs, the longer the computation will be, but on the other hand, systems with low connectivity (number of springs per particle) tend to diverge more easily. In order to decide on a particular layout, some experiments have to be run then.

MSSs are the favourite tool in cloth modelling, because, as [3] points out, cloth is not a continuum; rather it is an interlocking network of fibres. Also, cloth is best modelled as a surface, a 2D model, which however is not appropriate in the case of musculoskeletal modelling. Application of MSS on solid 3D models is less common, mainly due to the lack of physical accuracy. In [22], a deformable model for volumetric objects, based on the MSS, was devised. Several additional constraints are applied to the model to ensure the preservation of volume and surface area. Authors present convincing test results that prove the capability of their framework and the mass-spring systems in general to process quite complex scenes with multiple deformable objects in real time.

Due to its computational time efficiency, MSS is the chosen model for this thesis. The fact that medical applications require rather accurate results would speak in favour of some of the methods presented in section 2.2. However, interactivity was a major requirement for this assignment and physics based methods cannot provide it (moreover, other research groups involved in the VPHOP project are simultaneously developing methods based on FEM for cases when interactivity is not crucial). The exact mass-spring system that will be used is the one Zelený [29] developed last year for the VPHOP project.

3. Collision detection and response

Apart from the models themselves, there is another important topic when considering soft bodies and that is collision detection and response. The difficulties bounded with soft bodies stem from their complicated reactions to external influences. Proper collision detection allows the simulated object the actual interaction with its environment and therefore the introduction of those external influences. Regardless of the chosen model, a complete soft-body simulation framework must include the handling of collisions. The following text will describe several methods for collision detection (CD) usable for deformable models, point out the main differences between CD for rigid bodies and soft bodies and discuss the efficiency of those methods in context of musculoskeletal modelling.

There are two general types of CD – discrete and continuous. The continuous CD predicts the movement of the objects before each step of the actual simulation and checks whether some objects are about to collide. If so, the collision response (CR) part of the algorithm will modify the movement in such a way to prevent mutual penetration of the objects. Discrete CD takes a set of geometrical models as the only input and outputs couples of areas of these models that intersect each other. The movement information is not taken into account at all. Most usually, the input is a triangular or tetrahedral mesh and the output areas are therefore sets of triangles or tetrahedra. CR is then responsible for update of the geometrical model in such a way that ensures that the primitives no longer intersect. This means that the discrete CD detects the collision after it happens and "fixes" it, which may result in lesser accuracy.

It is clear that the discrete CD will always be faster than the continuous, because the movement prediction in continuous methods can be quite complicated when general non-trivial objects are considered. Its main drawback, apart from potential lesser simulation fidelity, is that if the movement of the object is too fast in relation to the discrete time step, the collision may not be detected. This effect is called tunnelling – in one time step, there will be an object moving towards another object at high speed and in the next time step, the first object will appear "behind" the second, passing through it. Still, with the simulation time step small enough, the discrete CD should suffice for the purposes of this thesis.

For the purposes of the particle model that will be used for this thesis, an additional problem arises, as the primitives of the geometrical model (namely the vertices of the triangular mesh) are not identical to primitives of the physical model (particles). The devised solution, which is more completely described in section 4.3, is using the particles themselves for the CD instead of the surface mesh. Therefore, in this case the input of CD is a set of spheres with various radiuses (particles) and the output is a list of pairs of intersecting spheres. Also note that self-collisions are often mentioned as a special case of CD. However, as the muscles are rather stiff objects with additional constraints in the form of bones and neighbouring muscles, they are unlikely to ever get close to a self-colliding shape. Also, the springs in the MSS should provide enough force to pull the particles away from each other thereby preventing self-collisions. For this reason, self-collisions will not be considered in this thesis.

The most obvious, brute force method for CD would be to test each primitive of one object against each primitive of the other object for intersection. However, that would obviously be too slow for real-time use in complex scenarios. That is why various speed increasing mechanisms were developed. Some of them will be described in the following sections.

3.1. Bounding volume hierarchies

Probably the most popular mechanisms for CD are bounding volume hierarchies (BVHs). The idea is to recursively subdivide the object of interest and compute a bounding volume for each of the resulting subset of primitives (triangles, vertices or in the case of the discussed mass-spring model, spheres). The bounding volumes are trivial geometric shapes that envelop all the primitives in the assigned subset. Examples of such are bounding boxes (axis aligned – AABB, oriented – OBB), spheres, discrete oriented polytopes (k-DOP) etc. (brief description of those as well as more examples can be found in [28]).

Then, when checking for collisions, the hierarchy of the potentially colliding pair of objects is traversed from top to bottom (top-down). During the traversal, the bounding volumes are tested for overlap on every subdivision level. If no overlap is found, the objects surely cannot collide. If it is, the algorithms traverse the hierarchy further, but only through the children nodes where an overlap was detected. Finally, when the traversal gets to the bottom level of the hierarchy, i.e. to the leaf nodes, and still detects overlaps, the primitives stored in these nodes are finally tested for mutual intersection. The algorithm for recursive traversal of the hierarchy, written in C-like pseudo code, follows:

```
Traverse(BV a, BV b) {
     Define empty list of intersections L
     If (a and b overlap) {
          If (a and b are leaves)
               L = intersection test of primitives in a and b
          Else {
               For each children a[i] {
                    For each children b[j]
                         L = L + Traverse(a[i], b[j])
               }
          }
          return L
     }
     Else return L // returns the empty list
}
```

The speed up of this method stems from the computational simplicity of the overlap test, as the bounding volumes have trivial shapes. Also, far less tests are made due to the space partitioning. Moreover, for rigid bodies, this comes with almost no drawback, as the BVH can be computed in a pre-processing stage and during runtime it only needs to be transformed along with the object (if it is a moving object).

For soft bodies, the situation is more complicated, as the object changes its shape and therefore some primitives can get outside their bounding box, making the hierarchy invalid. As a result, the BVH has to be updated during runtime. From this arises the demand for BVHs that are able to be recomputed quickly. This is the reason why more simple BVs, such as axis aligned bounding boxes (AABB) or spheres are preferred over e.g. oriented bounding boxes (OBB), which are quite popular in rigid body frameworks.

Although they generally do not fit the primitives they bound as tightly as the OBB, they are faster to reconstruct.

There are generally two kinds of BVH update – refitting and rebuilding. Refitting does not change the hierarchy itself, i.e. all parent-children relations remain unchanged. Instead, the BVs of the individual subsets are what changes. Simply put, if it is detected that some primitive "got out" of its BV, the BV is enlarged. Rebuilding means that the nodes, that have invalid BVs are removed (along with their children) and the hierarchy is made anew. The rebuilding process is obviously slower – it does everything that is done during refitting, i.e. computation of BV, plus it has to determine where each primitive belongs. However, when a large deformation occurs, the refitted BV can become very large, which results in a very loose fit and therefore many "false positive" overlap tests. A good BVH for deformable objects should therefore combine both update methods, balancing their usage according to the extent of the deformation.

Van Den Bergen [25] states that his tests prove that refitting is not only ten times faster then rebuilding, but also sufficient as a sole update method for BVH using AABBs for deformable models. He admits that for radical deformations *"such as excessive twists, features blown out of proportion, or extreme forms of self-intersection"* there is increased overlapping of the boxes. But for deformations that do not alter the topology of the object, there is no significant performance loss.

In the field of cloth simulation, Mezger et al. stated that advanced numerical solutions allow the usage of large time steps for the simulation, which on the downside introduces tunnelling [15]. Suggested solution is to inflate the BV so that it accounts for the movement in the next time step. Therefore, they obtain information about both collisions and proximity and the CR mechanism work with this information. This method can be considered a hybrid between the discrete and continuous approaches.

Larsson and Akenine-Möller proposed [12] a very robust BVH for deformable objects that even accounts for topological changes (tearing, cutting etc.). They introduced an update mechanism that uses both refitting and rebuilding. Their collision handling consists of two phases – Update and CD. The update phase makes use of assumed temporal coherence. The nodes that were used in previous CD query ("active" nodes) are likely to be involved in collisions again, therefore the deepest active node is refitted and then the refitting continues bottom-up by merging the child boxes. To account for possible big deformations or even topological changes, the nodes are also "invalidated" during the update phase. This is done by comparing the volume of the parenting box to the sum of volumes of the children boxes. If the ratio r in equation (3.1) is lower than a given threshold, the relationship is considered as invalid and the children nodes are deleted. In (3.1), V_p is the volume of the parent, V_i is the volume of the i-th child out of k children total.

$$r = \frac{V_p}{\sum_{i=0}^{k} V_i} \tag{3.1}$$

The authors of [12] state that 0.9 as the threshold value for r yields good results. Further on they admit that for some degenerate cases this can result into a node being evaluated as invalid every step, forcing unnecessary rebuilds of the hierarchy. However, for common

scenarios the simplicity of the test means faster processing and mechanisms that would take care of the degenerate cases would actually result in slowing the whole algorithm. Apart from the nodes invalidated by the use of formula (3.1), nodes that were not marked as active in the last CD are also invalidated.

Even after invalidation of a node, its children are not rebuilt immediately. Instead a lazy rebuild scheme is used, i.e. the nodes are not subdivided until they are needed during the second, CD phase of the algorithm. This phase is a regular CD as described at the beginning of this chapter, only with the necessity to subdivide the given node if needed. Also, it is responsible for marking every visited node as "active" for the upcoming update phase. The proposed solution is very robust, fast, relatively easy to implement and the fact that it is a result of a long-term research gives it additional credibility. This makes it a good choice for frameworks working with complex scenes where "anything can happen".

Actually, any BVHs are generally a good solution for complex scenes, when compared to approaches outlined in sections below. They are easily used for self-intersection tests; they can handle rigid vs. soft body collisions in the same manner as soft vs. soft body, which is quite convenient and they are usually easily implemented. Obviously, with generality comes the drawback that some other methods, devised specially for a given purpose, can yield faster results.

3.2. Distance fields

Distance field is a volumetric structure that expresses the closest distance to a given surface from every point in the space which is covered by the field. Figure 3.1 shows three slices of distance field for a 3D object. The basic idea of CD for object that use distance field is to test vertices of one object against the distance field of the second object and if the distance is not positive, a collision is reported.

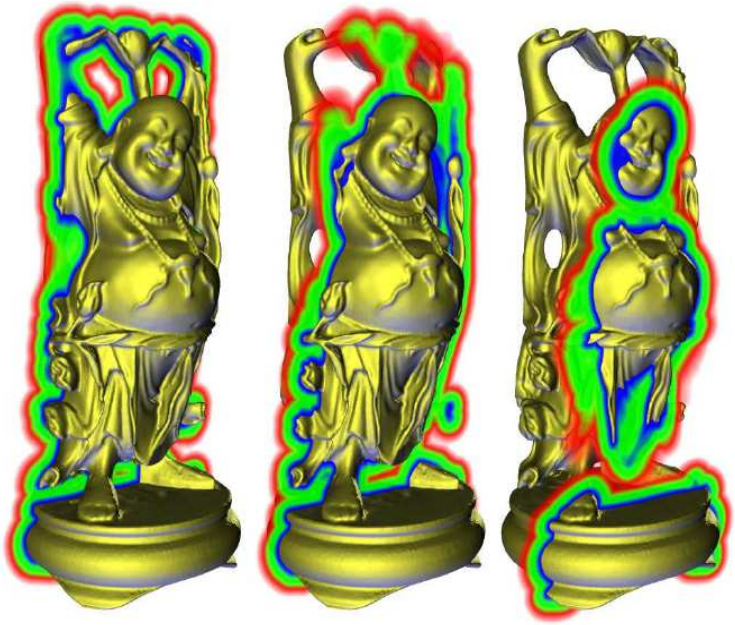

Figure 3.1: Three slices of distance field for the "Happy Buddha" model visualized using blue, green and red colours for closest, close and distant parts of the space respectively and fading out in the largest distance. Image taken from [24].

As [24] states, efficient computation of distance fields is still a challenging problem. This however poses a problem, as it is clear that distance field of a deformable object would have to be modified in every simulation step to be accurate. Although some attempts to incorporate distance fields for deformable objects were made, such as [6] or [4], they are far from real-time speeds. Although distance fields have some additional qualities, such as easier proximity queries [6], they do not seem to be fit for deformable models as well as for rigid. However, they are suitable for many applications in Computer Aided Surgery or similar fields, when the soft body collides with a rigid tool (e.g. organic tissue with a scalpel), where their performance could surpass BVHs. Such applications are not the subject of this thesis though and that is why they will not be described any further. Interested readers will find a more elaborate description and references to various other sources in the survey [24].

3.3. Spatial subdivision

Unlike BVH methods, which are based on the subdivision of the object at hand, spatial subdivision methods subdivide the whole R^n space and the fit the object into the created subdivision. Teschner et al. [23] used hash table for the spatial subdivision. Their framework works with tetrahedral meshes, although it could be modified to handle other primitives as well. The algorithm detects collisions in two passes. In the first, a hash value is computed for all the vertices of all the objects and they are assigned to appropriate cell of the table. In the second pass, hash values are computed for the minimal and maximal point of an AABB box of each tetrahedron. Intersection test is then performed for vertices in every bucket of the hash table, which falls in between those hash values. This means that unlike BVH methods, this method is dependent only on the number of primitives, not the number of objects.

The two passes described above have to be repeated each iteration to accommodate for the movement and deformation of the objects. The presented results show that the approach can process one iteration of CD in 70 ms for 20 000 tetrahedra, which would mean approximately 14 collision tests per second. However, bear in mind that this does not account for the time needed for the actual simulation of the underlying physical model. Unfortunately, the authors did not present times for the whole experiment.

In a more recent publication [14], similar approach is used. Instead of hashing each primitive, authors build a BVH consisting of AABBs on every object and hash the AABBs from bottom level of the hierarchy into a hash table. This way, far less hash value computations is made, as only the vertices of the bounding boxes are hashed, not all vertices in the scene as in the [23]. The collision tests are then made for each pair of neighbouring grid cells, i.e. for bounding boxes stored in the buckets of the hash table, which belong to those grid cells. Authors claim to achieve forty to seventy frames per second for one thousands up to fifteen thousands faces total for all the objects in the scene. The quality of the simulation is not addressed in either one of the papers though.

4. Solution design and implementation

This chapter will describe the designed solution that was implemented as a part of the "LHPBuilder", which is used as a unified software equipment for the part of the VPHOP project to which this thesis belongs. LHPBuilder is an application developed using the openMAF framework, an open source "Multimod Application Framework" [2], designed for applications based on the VTK (Visualization toolkit) [27]. This fact slightly influenced the design on some parts, i.e. the libraries available in the VTK were used when possible and also some other code from the VPHOP's framework was utilized. The framework uses C++ as the native language and Microsoft Windows as the platform.

4.1. Pipeline of the method

The overall pipeline of one simulation step is captured by Figure 4.1. The first notable fact apparent from the figure is that the solution uses a mass-spring system (particle system) as the soft-body model. Details about it can be found in section 4.2. The pre-processing phase prepares the input for simulation for both the MSS as well as the collision handling. Therefore, the actual operations done in this phase will be described partly through all the following sections of this chapter.

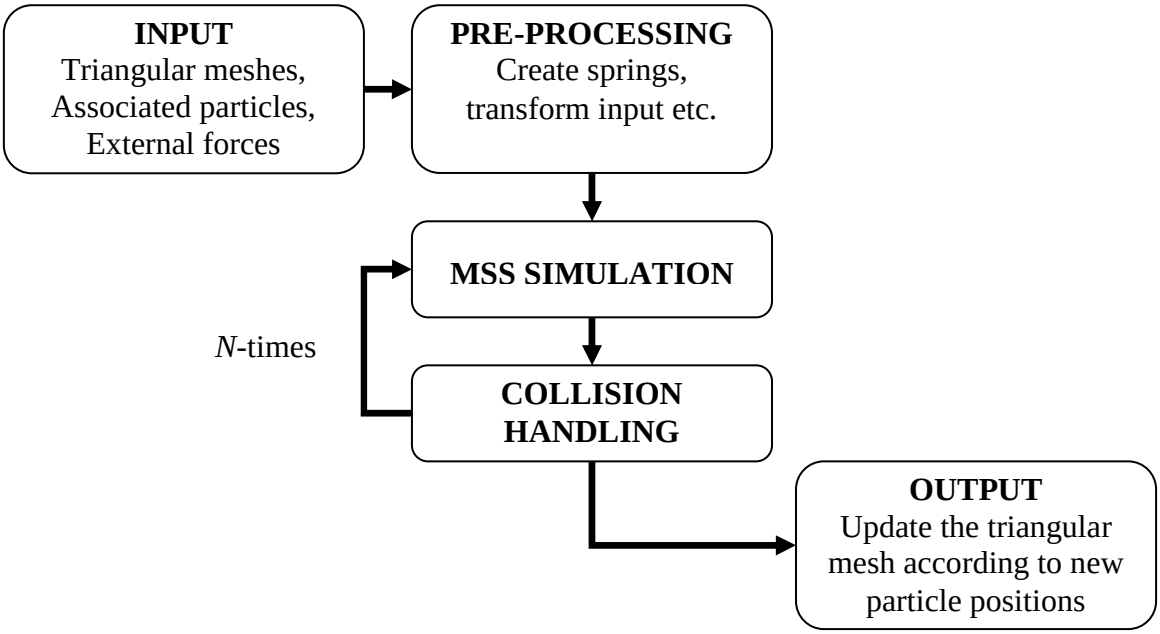

Figure 4.1: Pipeline of a soft body simulation using mass spring system.

One more notable thing in figure 4.1 is that once the simulation begins (i.e. after pre-processing phase), the system does not account for any explicit outer influences. This means that any rigid movement and forces that are supposed to be applied to the objects have to be defined before the simulation begins (during pre-processing).

The whole simulation covers certain amount of time, which is discretized into a number of time steps. Each of these time steps can be divided into two parts – rigid-body simulation and soft-body simulation. During the rigid-body simulation part the rigid objects (bones) in the simulated scene move. The rigid movement can be based on some prescribed pattern or on user's interaction and can be easily described by geometrical transformations of the

objects. The soft-body simulation is then used in each of those steps to generate a proper shape for the soft bodies in the scene, while the movement of rigid objects provide the external forces influencing the soft bodies. Therefore, the process depicted in figure 4.1 is repeated whenever the rigid objects move, i.e. each time step of the simulation.

The LHPBuilder allows the user to move through the timeline arbitrarily, implying that two consecutive simulation steps do not have to be continuous. The user can even go backwards in time, remove or add objects into the scene or change parameters of the simulation in between the time steps. Although this means complete freedom for the user, it comes at a price. Because no assumptions can be made about the input or the time causality, the pre-process phase has to be executed in every step of the soft-body simulation and some of the data structures created in that phase are again removed at the end of the step, because there is no point in keeping them. All the applied movement transformations are always in respect to the rest pose of the objects. This means that even if the user goes through the time steps in an ordinary time succession, the program will move all the objects from rest pose to the current time pose in each step, building and destroying all needed structures in the process.

This has a negative impact on the overall performance. Should a future implementation of LHPBuilder contain some kind of “animation mode”, i.e. such mode in which the user would set the parameters and then just “hit a play button”, it would be useful to build a specialized pipeline for this mode. It would keep all the data structures, such as bounding boxes for collision handling or initialized MSSs of the soft-bodies, and the simulation in each time step would always continue from where the previous step ended. Nevertheless, current implementation does not account for such possibility and treats every simulation step as an isolated simulation. This approach will be assumed through the remaining text.

When the data are ready, the simulation begins. One iteration of the simulation consists of two passes – mass-spring system iteration and collision handling. Please note the difference between one simulation iteration and one MSS iteration. One iteration of the MSS means integrating one infinitesimal time step dt (as in equation (2.14)).

The number of iterations N can be set according to different criteria. One possibility is to measure the changes in the output (e.g. average vertex displacement) between iterations and stop the simulation once the changes are none or at least small enough, as in general case the system does not have to converge to a standstill state. There are two reasons for this: first, the MSS might oscillate and second, as the muscles are almost in permanent contact, there will almost always be some collision which will make some particles move and therefore will break the equilibrium of the MSS. Another possibility is to have fixed number of iterations, based either on the amount of simulation time that should be integrated, or possibly on experimental results, e.g. using an average amount of iterations that was needed to achieve the “sufficiently stable” state when using the first method. Third method can be employed when the speed of the simulation is crucial – instead of iterating N-times, the loop can be changed to iterate until assigned time window was depleted and the output must be passed on. While the first method will produce the best result in terms of quality, the last method will obviously be fastest. As one of the main requirements for the designed solution was speed, the first method is not very suitable. On the other hand, the last method is very restrictive and not suitable for testing purposes. Therefore, the current implementation uses the compromise that is the second method (i.e. fixed number of iterations based on experiments, see section 55.4 for details). However, it

can be easily changed if future requirements demand either higher deformation fidelity (first method) or speed (last method).

The last section of the pipeline is generating the output. At the beginning of chapter 2, it was mentioned that the muscles in the musculoskeletal model used in the VPHOP project are described by two models. Figure 4.1 implies that the designed solution works with both the volumetric (fibres) and surface (triangular mesh) model of the muscle. As was already mentioned in section 2.3, the particles used in the MSS are obtained by sampling the muscle fibres, which are – implementation-wise – polylines. This makes the deformed muscle fibres easy to obtain after the soft-body simulation, simply by connecting the particles in the same way they were connected by the initial fibres. To obtain the deformed surface model, some correspondence between the particles and the initial model have to be established. Section 4.4 contains description of how this is done, along with other information about how are the input and output data specified and handled.

4.2. Soft-body model

It was already established in section 2.3 that mass-spring system will be used for the model. Moreover, there is already a MSS implemented in the LHPBuilder, therefore it was used instead of implementing it anew. This covers the computational core of the model, i.e. the solver that computes new positions of the particles. There are still several possibilities on how to choose the spring layout, parameters and other settings of the model though.

First, the computational core of the MSS will be described briefly. It was designed and implemented by Zelený [29]. Although Zelený designed a fast parallel GPU version of the MSS, only the basic non-parallel CPU version is present in LHPBuilder, because the GPU version was not optimized for data sets that could not fit into the GPU's memory and this problem have not been resolved yet. It takes an array of particles (3D points) and springs (one spring is defined by end points, stiffness and rest length) as the input. The size of the time step *dt* can be chosen as well as the damping coefficient of the springs (only homogenous materials are considered, therefore the coefficient is the same for all). An important action the implementation allows is to set "fixed" points. The positions of the particles that are marked as fixed are not changed by the solver.

The fixed particles are utilized for representation of attachment areas of the fibres, i.e. the areas in which the muscle is "fastened" to the bone. After the particles are sampled along the fibres, the particles closer than a chosen threshold to some bones are declared as fixed (see 4.4 for more details). During the soft-body simulation, those particles propagate the movement of the bones into the MSS in the following manner: it was stated in section 4.1 that in every time step of the simulation, each bone is assigned a certain geometrical transformation. The pre-processing phase of the soft-body simulation applies these transformations onto the particles, which were fixed to the appropriate bone. As these particles cannot move, in order to achieve an equilibrium state, the unfixed particles of the MSS will be forced to assume new positions. This way the muscle will acquire a new shape that will reflect the movement of the bones. In the following simulation step, the bones will have different transformations assigned (if they are actually moving) and the process will repeat.

In fact, every particle is transformed at the beginning by the same transformation as their nearest bone, even those that are not fixed. This way the particles start the simulation in positions which are much more close to the final positions, therefore the MSS will stabilize

more quickly than if the particles would have to start at the untransformed rest pose. Moreover, the rest pose may be very distant to the current pose. Although the springs would force the unfixed particles to go the appropriate position, they could collide with some objects that would "stand in their way", even though this collision would never happen in real life scenario. The transformation of all particles solves this problem.

To set the parameters of the MSS, the rules proposed by Zelený and already described in section 2.3 were used. Concerning spring layouts, several approaches were tested. Section 5.1 contains results of these tests and concludes which fits the application best. The model does not account for gravity, friction or similar influences. Although the radiuses of the particles vary, the mass is the same for all of the particles – the varying radiuses are used solely for the purpose of collision handling.

4.3. Collision handling

The collision handling mechanism is responsible for making sure that no objects in the scene penetrate each other. However, section 4.2 established that the surface of the objects is not used as a basis of the soft-body model. In order to use the surface model (triangular mesh) for CD, it would mean that it would have to be updated in each iteration of the soft-body simulation to reflect the changes of positions of particles. This seems like a wasteful operation, because the surface mesh is not needed for the soft-body simulation itself, it is sufficient to update it only at the end of the simulation and output it.

Moreover, CR would be complicated as well if the surface mesh was used. After colliding triangles would be found, they would first have to be transformed in such a way to remove the intersection. This operation itself is rather complicated, as there are numerous possible transformations that achieve this and it is not trivial to decide which one is the right one. Even after resolving this problem, it would be necessary to somehow propagate those changes back into the particle model, meaning another update step, this time updating particles on the basis of the triangle model.

If the particle model itself was used, all the update actions would not be necessary, saving a lot of computational time. The problem is, though, that the particles are point masses, i.e. they have no volume. The springs as well are considered as infinitesimally thin. Any attempt to collide the particles of one object with another would yield no result then. And although they form the volumetric model of the muscles, they do not trace the exact shape of the surface. But the surface is needed for the CD – a collision is after all recognized by mutual penetration of the objects, which means that the surfaces of the objects intersect. What is needed then is a way to describe the surface of the object without actually using the surface mesh but only the volumetric particle model.

The suggested solution is to stop thinking about the particles as points but instead use them as spheres. By inflating the radius of particles that are close to the surface to such extent that they actually touch the triangular mesh, they can approximate the surface. And if there are enough particles, most of the surface will be covered by the spheres. Then, instead of testing the triangular meshes, the "sphere meshes" would be tested for collisions.

For the CD mechanism that should be no problem – the primitive type changes from triangle to sphere, but most of the CD approaches are not affected by that. The piecewise test for collision is actually simpler for spheres than for triangles – if the sum of radiuses of two spheres is larger than the distance of their centres, they collide. What is more, the CR

becomes very simple as well: it simply moves each colliding particle away from the other by half of the difference between their current distance and the sum of their radiuses. Following pseudo covers these two operations more exactly, while figure 4.2 provides additional guidance.

```
Piecewise sphere CD and CR (Sphere A, Sphere B) {
     V = A.centre – B.centre
     Distance = Length of V
     If (Distance < (A.radius + B.radius)) {
          Difference = (A.radius + B.radius) – Distance
          Vd = normalize(V) * (Difference / 2)
          A.centre = A.centre + Vd
          B.centre = B.centre - Vd
     }
}
```

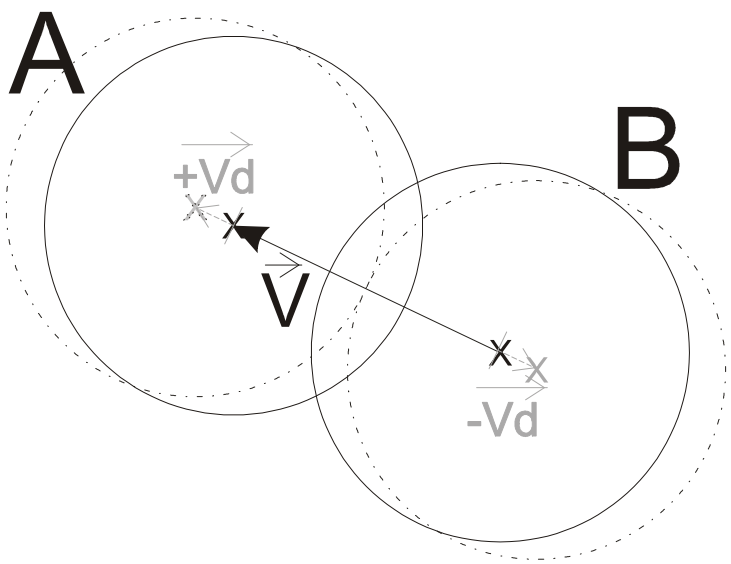

Figure 4.2: A schematic for collision response of two spheres A *and* B *(projected into two dimensions). The solid lines mark the initial colliding state, the state after the response is dashed. The* V *is the vector obtained by subtracting the centre of B from centre of A.*

In real scenarios, one particle may collide with multiple other particles. To account for such situations, the V_d vectors are not added/subtracted immediately when the collision is found, but they are stored and accumulated for each collision of the given particle. When all collisions are processed, the superposition of all the accumulated vectors V_d, which simulate the force the colliding particles applied, is added to the position of the particle. This way, all the collisions of the given particle are tested against the same initial position of that particle. The collision counter is also utilized as a marker of which particles had a collision. The movement of these particles is stopped by setting their previous position to the same as current position, effectively giving them zero speed. This means that the particles do not rebound from each other at all after collision.

To sum up, the suggested method is actually simpler (and therefore faster) than if triangles were used and it changes the positions of the particles directly, making it even faster. The obvious drawback is that it is not as exact, because the spheres will not cover the whole surface perfectly and therefore some minor penetrations can occur. Nevertheless, the overall performance of the method should be sufficient in terms of quality and much more efficient in terms of computational time than if the "conventional" triangle meshes were used.

To compute the radiuses of the particles, the closest particle to each vertex of the surface mesh is found. Note that one particle can be the closest one to several vertices. Also, only the particles that are on the "boundary" fibres, i.e. fibres closest to the surface, have to be accounted for – the internal particles surely will not collide with other objects (if they do, the simulation is surely flawed). After this relationship is established, the radius can be set so that it touches the most distant of the points to which it is closest. This way, the largest portion of the surface is covered. However, this also means that the particles cover a lot of space which is actually outside the boundary of the object, therefore a lot of collisions can occur which should not occur. It is obvious that the smaller the radius is the lesser portion of both the surface and the "excess" volume is covered. To make a compromise between the two variables, the radius of each sphere is set to an average of the distances between the centre of the particle and the associated vertices. This can generate some very large particles, because some vertices are very distant to any particle. Therefore, the CD mechanism is used to detect particles that intersect and then the radiuses are decreased so that the intersection are removed, which effectively removes the very large particles. To increase the coverage of the surface without increasing the volume excessively, more particles per fibre can be used. That, however, comes at a price of higher memory and computational time demands.

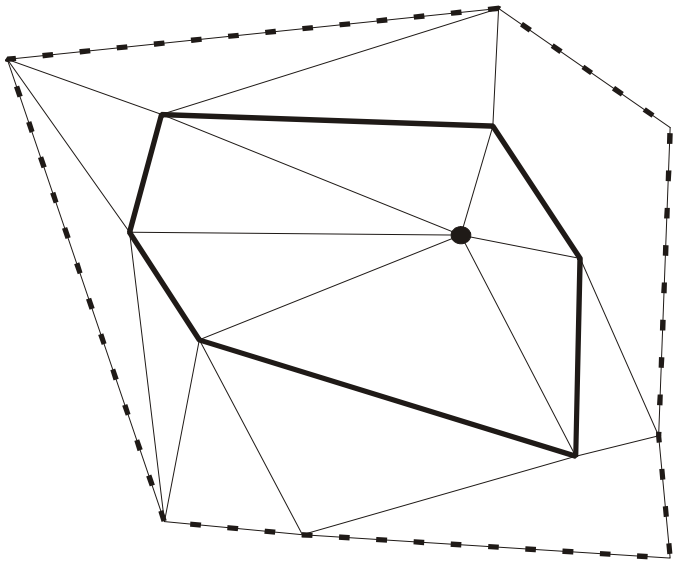

Figure 4.3: A part of triangle mesh. Thick solid line connects the vertices of one-ring of the root vertex marked by black dot; dashed line connects the vertices of its two-ring.

The solution devised above would work only for soft-body vs. soft-body collisions. When soft-body vs. rigid-body CD is required, there arises a problem – the rigid object does not have any particle model, it is represented solely by the triangular mesh. One possibility

would be to use sphere vs. triangle collision tests, which would be a little bit slower than sphere vs. sphere and also it would require additional programming. Therefore, spheres that approximate the surface of the object are generated for the rigid bodies as well. It is actually easier for the rigid objects than for the soft-bodies, as the number of spheres and their position can be arbitrary, as they server only one purpose. There is a theoretical flaw of this method in being less accurate then if the sphere vs. triangle test were made. However, the spheres of the soft-body only approximates its surface, therefore the accuracy of testing it against the actual surface of the rigid-body is disputable.

The goal is to have the surface of the bone covered as much as possible while having as few primitives as possible. The suggested solution is to generate one sphere for each disjoint set of triangles defined by a ring of vertices, i.e. one “root” vertex and its neighbouring vertices (see figure 4.3). The longest edge of all edges passing through the root vertex is then found and so is the vertex normal vector (as a weighted average of adjacent face normal vectors). The centre of the sphere, which’s position is initially equal to that of the root vertex, is moved against the direction of the vertex normal by the length of the longest edge, i.e. “inside” the object. The radius is then set in such fashion so that the sphere touches the ending vertex of the longest edge, i.e. to the length of the edge multiplied by a square root of two (diagonal of a square). This heuristic was chosen as a compromise to trace the surface as closely as possible while not covering much of space outside the surface. To sum up, the algorithm starts with one vertex, create a sphere based on the neighbouring vertices and marks all the involved vertices. Then it continues with the remaining vertices of the mesh the same way, but always skips the marked vertices to ensure that the processed triangle sets are disjoint.

Note that the heuristic based on the longest edge works best when the mesh consist mostly of regularly shaped triangles with equally long edges. If some degenerate triangles appear in the mesh, it might result in a very large sphere that might consequently result in erroneous CD. There are meshes of different quality available in the data sets used in LHPBuilder for each bone. The most refined meshes, i.e. the ones with most triangles, are used when building the sphere representation of bones. They contain very fine triangles and therefore large quantity of vertices. To ensure that there is not an excess of spheres generated, a two-ring (see figure 4.3) of vertices is used instead of one-ring. Also, semi-random skipping of several vertices can be employed to reduce the number of spheres. Figure 4.4 shows a result of this approach. While the initial one-ring design ensures that the whole surface is covered, it actually generates a lot of overlaps among the spheres. Also, much less spheres are generated with the randomized two-ring approach, which is why it is the suggested option.

Not only does this allow the collision testing between both types of objects, but the same mechanism can be used to do so. This makes the resulting code more refined and easier to comprehend. Both soft-body and rigid-body objects use the same structures, except that the MSS related data are not generated for the bones. On the first sight, it might seem that the CR must be a bit different, because the bones should not be affected by it – they do not move. Therefore, the whole difference of the sum of radiuses and actual distance of the colliding spheres has to be added to the particle of the soft-body instead of adding half of it to each of the particles. However, the same rule has to be applied in some cases even during soft-body vs. soft-body collision and that is when the fixed particles are involved in the collision. This means that there is no difference in the handling of both types of collisions after all – the bones only have to have all the particles set as fixed.

Out of the algorithms presented in chapter 3, two were a suitable choice for the CD – either the BVH approach described in [12] or the spatial subdivision approach from [14]. Both promised fast results and both are able to handle input data as specified in the text above. The former was chosen in the end due to seemingly simpler implementation and higher credibility (see sections 3.1 and 3.3 for description of both approaches).

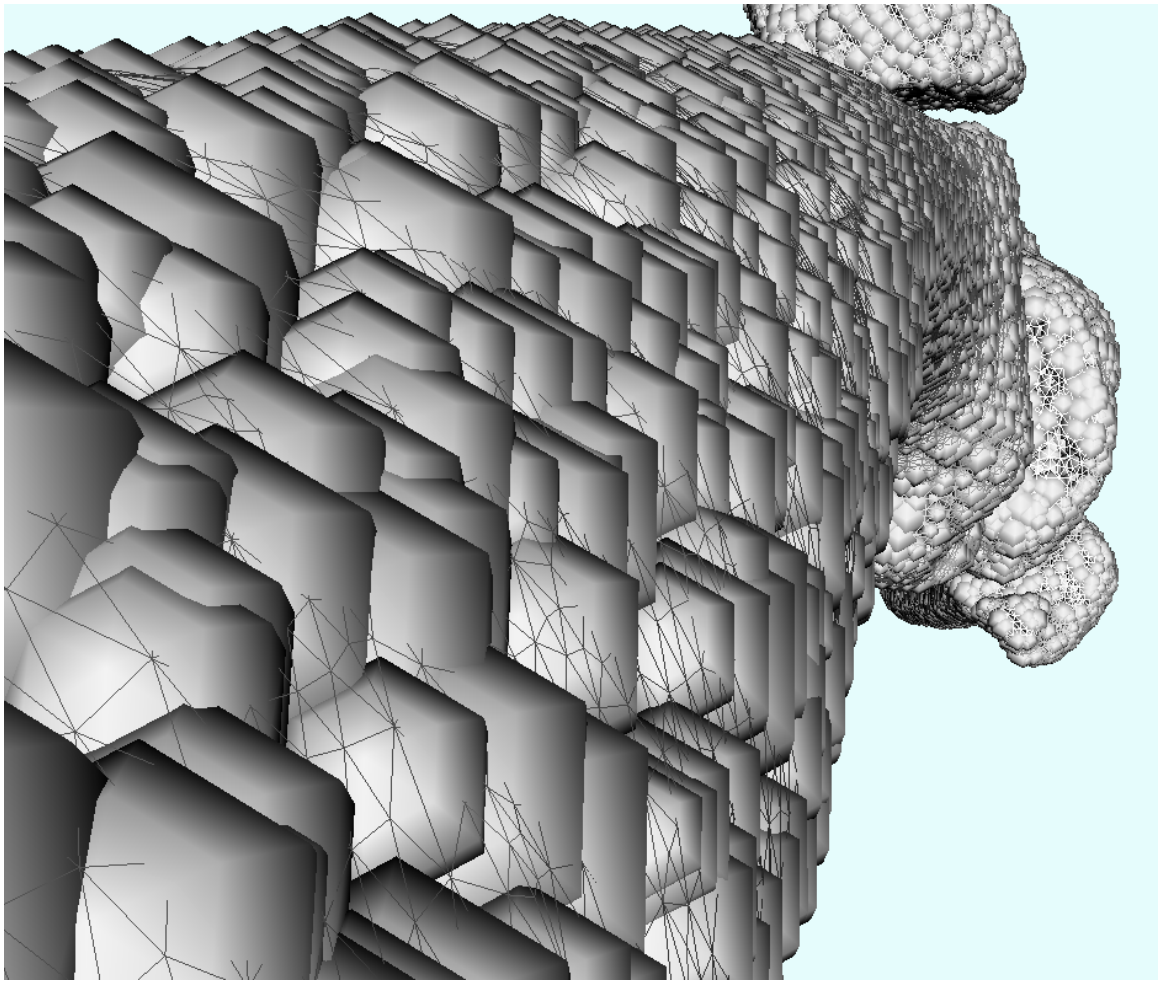

Figure 4.4: Visualization of the surface model (wireframe triangles) of a thigh bone and knee approximated by a set of spheres (solid).

The chosen method uses dynamically built bounding volume hierarchy. The bounding volume chosen for the implementation was a simple AABB, for reasons that were already declared in section 3.1. The bounding boxes are subdivided into octants in each level of the subdivision. The division lines always pass through the midpoint of the parent box. Whenever new object is added to the scene, the bounding box is constructed for it on the parent level, i.e. without any subdivision.

At the beginning of the collision handling step of the soft-body simulation, the BVH of each object is updated. The update procedure takes the current number of the simulation iterations (MSS simulation + collision handling, see figure 4.1) to decide which parts of the hierarchy are too old (any which were not used in the previous iteration). The update proceeds with refitting and rebuilding according to the rules described in section 3.1. A slight exception is for the bones. Their bounding boxes never need to be updated, because the rigid objects do not move (not in the span of one simulation step) or change shape.

Then each muscle in the scene is tested against each other muscle and also each bone. The BVHs of the objects are traversed and subdivided when needed, while marking each visited node with the iteration number (for the purposes of the update phase). The implementation allows choosing maximal number of recursion steps as well as minimal number of primitive per node. Once either of the limits is reached, and there is still a collision detected between the nodes, a pair of lists containing the primitives in the two colliding nodes is outputted. The piecewise test of the primitives in these lists is then done outside the code of the BVH class. After all tests are finished, the results are applied (i.e. the accumulated vectors V_d are used to update the positions) and a new iteration may begin.

It was stated in section 4.2 that all the particles always undergo the same transformations as the bones to which they are close. Also, the same bounding box is always used only for one time step of the simulation (see section 4.1 for the reasons). This means that the scale of expected deformations actually is not very large. Therefore, it is worth a try to test Van Der Bergen's claim, mentioned in section 3.1, that the rebuilding of BVH is usually not needed unless the deformations are vast. The version of CD with and without rebuilding was tested and the results are provided in section 5.2. This is also one of the reasons why discrete rather than continuous collision handling mechanism was chosen (apart of simpler implementation and faster execution). Because the deformations should not be vast, the particles should move only over short distances and rather slowly, therefore discrete CD should be sufficient.

4.4. Input and output data

4.4.1. Raw input data

The designed solution uses several types of data objects. First, it is the surface models of both soft and rigid bodies in form of triangular meshes. The vtkPolyData class from the VTK libraries is used for this purpose. These meshes were obtained from a medical scan of a real human. As the output of such scanning methods, such as MRI, usually produces a lot of triangles for each mesh, techniques for mesh decimation were employed to produce several meshes of each object with different primitive count. For the muscles, the user chooses which mesh will be used. For the bones, the original, finest meshes are used (see section 4.3 for reasons).

The particles are not present in the input data set (at least not in the current implementation), they have to be created. The muscle fibres, along which the particles are generated, are specified as polylines (vtkPolyData is used for that purpose). The user selects the number of particles per fibre and the number of fibres. Then the particles are generated on each fibre with equal distances between each other. "Neighbour" relations between the particles are generated according to a selected spring layout and stored as well. These relations simply mark which pairs of particles are to be connected by springs. Note that the particles are built only for the rest pose fibres for the reasons stated in section 4.1. Therefore, they are generated only once per the whole simulation, unless the user changes some crucial parameters (e.g. when desires more particles per fibre). After the particles are generated, the fixed particles are found and marked. Each particle which is closer than user specified threshold to some bone is marked as fixed to the bone.

To improve how well the particles cover the surface of the object for the purposes of CD (see section 4.3 for details), the position of the second generated particle on each fibre is generated randomly in the interval <0; *SL*> (the first should remain at the beginning of the

fibre in order not to shorten it), where *SL* is the length of the sampling segment, i.e. the distance between each particle on the fibre. The remaining particles are then sampled as usual, in length intervals equal to *SL*. This way, the particles are more scattered inside the muscle, which is better for the devised CD mechanism. Figure 4.5 documents both situations. Also, figure 4.5 shows that in the real data, some parts of the muscle are not covered by the fibres and therefore by the particles (the protrusion in the lower part of the muscle). No collisions will obviously be detected there and therefore undetected penetration by some other object may occur there. This is considered as imperfection of the data set and therefore not solved in this thesis.

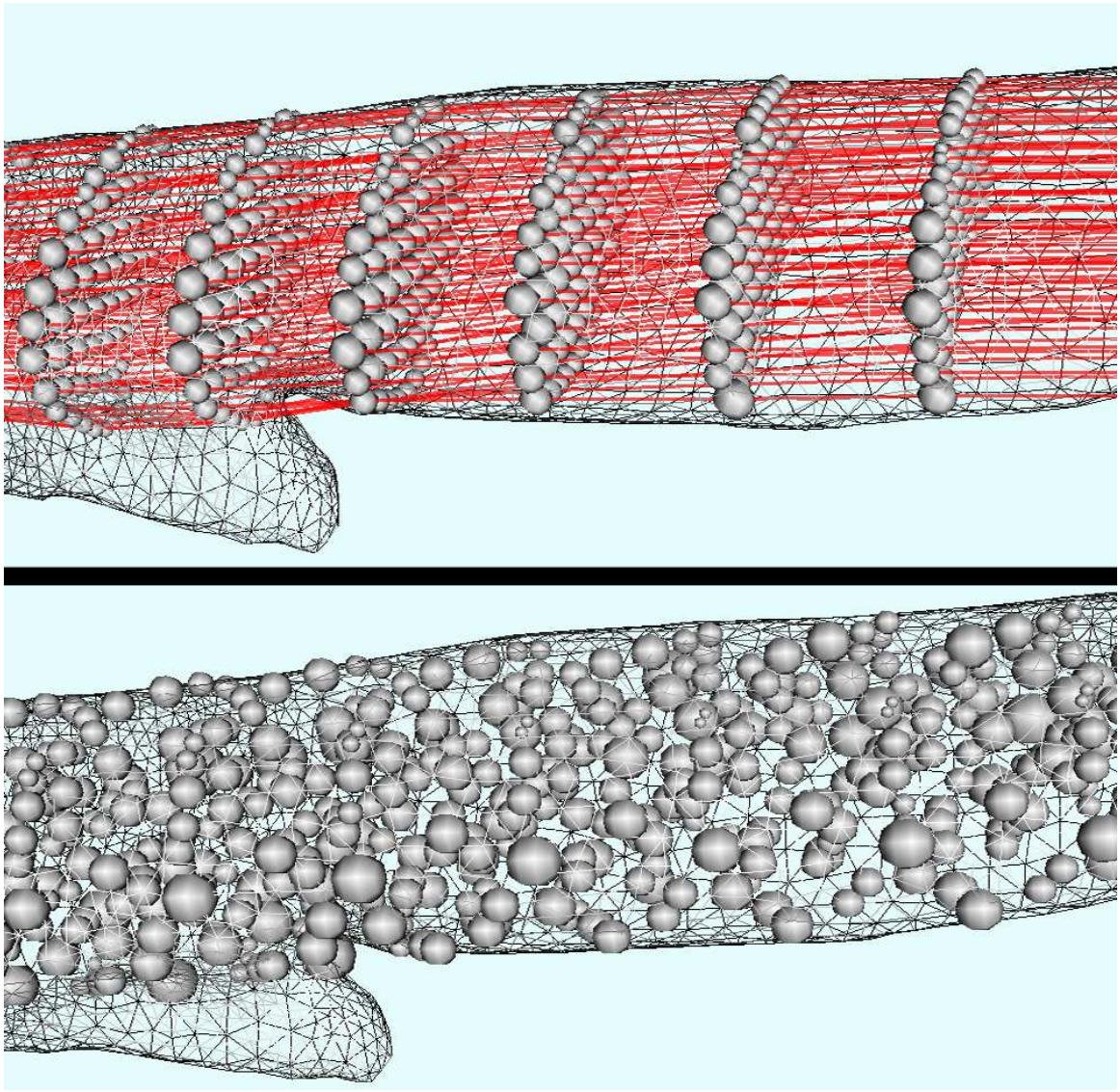

Figure 4.5: Sampling of particles along muscle fibres without (upper) and with (lower) randomization. The surface mesh is depicted as a wireframe model. The muscle fibres are displayed only on the upper image (for better clarity) as red lines.

4.4.2. Refined input data

The surface meshes and the particles can be considered as “raw” data, which have to be further processed in order to be of any use. Structure vtkMSSDataSet (see appendix for more details on the source code) was designed to hold the refined data that are obtained

from the raw data. Section 4.1 stated that because of how the implementation is designed, most of the input data have to be built anew in each simulation step. An instance of the vtkMSSDataSet contains all the data that are reusable, therefore it has to be created only once per object per the whole simulation, unless the raw data change.

The data stored in the vtkMSSDataSet are following: positions of particle centres, boolean arrays indicating fixed and boundary particles, radiuses of the particles, an array of indices of bones to which a given particle is fixed, springs connecting the particles and an array containing a specified amount of closest vertices of the surface mesh to each particle. The following paragraphs will describe how these data are obtained and what are they used for.

How the positions of particles are obtained was already discussed and so was the obtaining of the fixed particles descriptors. The indices of bones to which the particles are fixed are obviously stored at the moment when the fixed particles are being marked. Most of the other structures stored in vtkMSSDataSet are generated using the particle positions. The radiuses of the particles are crucial for the CD. How exactly are they used and how are they generated is therefore described in section 4.3.

Several options on how to generate the springs (i.e. several spring layouts) were tested and their differences are shown and discussed in section 5.1. For layouts based on cubical lattices, the springs are set as the edges of the template lattice (see section 2.3). Another tested layout is based on Delaunay tetrahedralization. The tetrahedonization is built from all the particles using the VTK library class vtkDelaunay3D. The springs are then the edges of the resulting tetrahedra. The last method for layout creation that was tested uses *N*-closest particles. In this layout each particle is simply connected with a specified amount of closest particles.

The spring is defined by its endpoints, i.e. by indices of particles. It is assumed that the initial position is indeed the rest pose of the model, therefore the rest length of the springs is set as the distance between its endpoints. The stiffness of each spring is set as inverse of the rest length (see section 2.3 for reasoning).

The remaining structure that is constructed is the array of closest particles to each vertex of the triangular mesh. It is constructed using a brute force method (as this is done in the pre-processing stage, the computational time is not crucial), simply by finding *N* closest particles to each vertex among all the particles. This structure is used to update (deform) the surface mesh once the soft-body simulation ends. Its detailed functionality is described in section 4.4.4.

4.4.3. Temporary data structures

Once the refined data are prepared and the soft-body simulation is called, another set of data structures has to be prepared. These are the data structures that last only during the one soft-body simulation step and are discarded in the end. They are the mass-spring system itself and the hierarchical bounding box used for collision detection.

The MSS was described in section 4.2. It takes the particles (only their positions, not radiuses) and the map of fixed particles as its input. During the simulation, the MSS computes new positions for every particle in each of its iterations. To ensure reusability of the input positions of particles in the next simulation step, the MSS therefore has to make its own copy of the particle positions, which it then modifies during the simulation.

The collision handling mechanism obviously has to work with the new positions the MSS produces and in return, the MSS needs the position after the collisions were processed in each iteration. This is resolved very efficiently – both the MSS and the BVH share the data structure, i.e. the pointer to the array with particle positions, and therefore they both write into the same array.

The BVH's only input are the positions of particles and it shares those with the MSS, therefore the only memory it allocates and clears is that needed for the nodes of the hierarchy itself. The MSS is responsible for deallocation of the particle's position array after the end of the simulation.

There is a class, designed as a VTK filter, named vtkMassSpringMuscle, which represents the muscle for the duration of the soft-body simulation. It encapsulates the two structures discussed above and provides methods for all the needed actions – pre-process, simulation and output dispatching.

In the pre-process phase, it creates both MSS and BVH, creates copy of the input mesh (so that the input is not modified and can be reused in next simulation steps) and transforms the refined input data by appropriate transformations. During the simulation itself it is responsible for the piecewise collision tests of the primitives detected by the BVH. After the simulation, it prepares the output (see section 4.4.4). Also, a vtkMassSpringBone class is inherited from it to represent and handle the bones in the same manner. All instances of these classes are disposed of after the simulation ends.

4.4.4. Output data

The update of the fibres is quite straightforward. It is kept which particles were initially on which fibre as well as in what order all particles on a given fibre were. To get the "new" fibres, all that has to be done is connect the particles by line segments according to the stored scheme.

The update of the surface mesh is a bit more complicated, because there is not any direct connection between the particle model and the triangular model. This is where the N closest particles to each vertex are used. When the soft-body simulation begins, the starting positions of the particles, i.e. after the rigid transformations are applied, are stored. Each vertex of the input triangular mesh of the muscle is also rigidly transformed by the same transformation as its N closest particles. However, as its N closest particles can each be bound to a different bone and therefore have different transformations assigned, the given vertex is transformed by an average of all those transformations. That means its initial position is transformed by each of the transformations of its closest particles and an average of these positions is set as the result (so if all the transformation are the same, the result will be the same as if simply one transformation has been made).

After that is done, the soft-body simulation is started. During it, the mesh is not updated at all. After it ends, a translation vector for every vertex is computed as an average of vector differences between the end positions and the stored initial positions of the N closest particles (a superposition instead of difference was tested as well, but average produced much more pleasing results). This vertex is then simply translated by the resulting vector. The quality of this method is further discussed in section 5.5.

One could argue that the rigid transformation of the mesh before the soft-body simulation starts is unnecessary as one could simply use the final positions of the particles and the positions in the initial pose (before the rigid transformations are applied) to compute the differences and then use them to translate the vertices of the mesh also directly from the initial pose to the final. Note however, that it might not be possible to describe the rigid transformations of the bones solely by translation (e.g. when some rotation is used to transform the bones). Therefore, the intermediate transformation step is indeed necessary.

5. Experiments

Various possible approaches to individual problems related to the proposed method were listed through chapter 4, such as what kind of spring layout should be used, how to update bounding boxes used for CD, how to choose the radiuses of particles etc. This chapter contains results of test that were made in order to test some of those approaches and compare them to each other. Moreover, overall performance tests for the whole method are presented. The chapter is divided into several sections, each containing results of one test, usually in forms of tables, charts and images, and a brief discussion related to the given test. The discussion clarifies why a certain approach, algorithm or parameter value was chosen for the final application. It is assumed that the reader is thoroughly familiarized with the content of the whole chapter 4.

The following tests were made using the LHPBuilder application, compiled for the Microsoft Windows / x86 platform. A usual desktop PC was used for the testing. Its configuration was: CPU Intel Core2Duo E6300 (two cores, 1,86GHz clock speed, 2 MB L2 cache, introduced in 2006), RAM DDR3 2GB (1066MHz), HDD Western Digital Caviar SE WD2000JS (Sata II, 8MB cache, 7200rpm), OS Windows 7 Professional.

One data set was used for all the tests. It consists of MRI footage of pelvis and legs, fused with a motion capture data of a walking human doing three steps[1]. The whole movement of all bones in the data is captured in figure 5.1. A total of twenty three muscles are available for testing in the data set, ten of which are on the pelvis and thirteen on the right thigh. Figure 5.2 shows the rest position of the musculoskeletal model. The rest position is the one to which all the used rigid transformations are related. The camera's view angle was set the same when taking both figures 5.1 and 5.2, so it is visible that the first position of the moving data is actually rotated along the vertical axis. The leg bones obviously have various different transformations applied and the whole moving model is also translated away from the rest pose model (that is why they are not captured on the same figure, it would not fit the page).

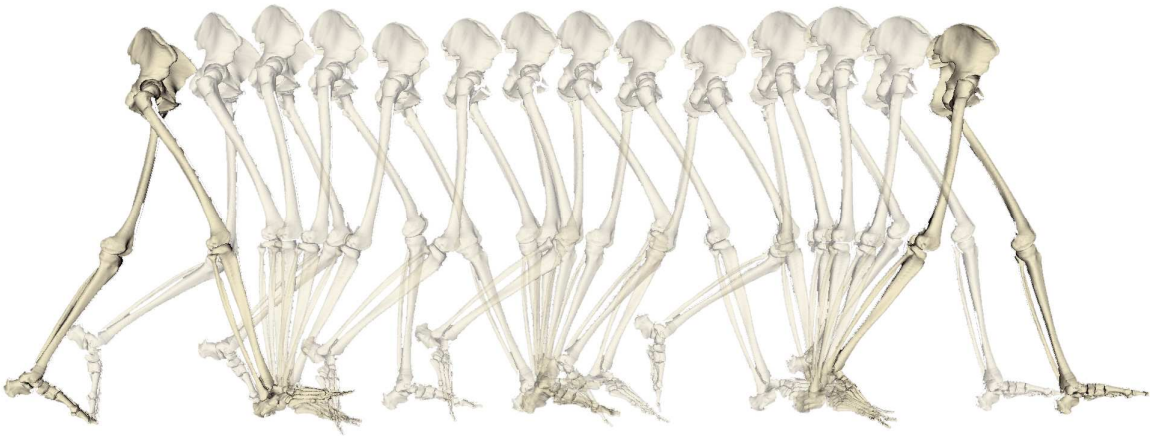

Figure 5.1: Fourteen frames capturing the movement of the musculoskeletal model (only bones are visualized) in the testing data set.

[1] This data set was created as a part of VPHOP activities.

Figure 5.2: The rest pose of the musculoskeletal model (only bones are visualized).

5.1. Spring layout

Number of iterations needed to achieve the final shape of the soft-body is what affects the computational time the most. The spring layout has a huge impact on this number, therefore several layouts were tested. The test processed 1500 iterations of the simulation for each tested method, measuring the average displacement of particles, i.e. the difference in position of each particle between two successive iterations. In ideal state, the particles would not move at all once the final position is achieved. However, this will almost never happen in the testing simulation. The model itself might oscillate and, moreover, the muscles are in almost permanent collision. In the position, which is the closest to the final position, the objects will most likely be caught in a loop – the MSS generates displacement based on the forces of the springs in order to achieve equilibrium, which results in some collisions. The collisions get resolved, the positions of particles change, which results in disturbance of the equilibrium and the process repeats. Rather then zero displacement, an oscillation of the displacement values is a sign of the final state.

Apart from the displacement, the overall computation time was measured. The layouts differ in the total number of springs, which slightly affects the time. The tested layouts were: Cubic lattice models with 6 and 26 springs per particle, Delaunay tetrahedralization and 15 nearest neighbours (see section 4.4.2 for their descriptions). The times for those methods were 1111.156s, 1296.031s, 1241.938s and 1213.703s respectively but please note that these times are listed only for comparison between different spring layouts and do not reflect the performance of the final solution, see section 5.6 for a complete analysis of time consumption. In the test, the first position of the moving data of all the right thigh muscles was simulated.

Figure 5.3a and 5.3b show the resulting displacement through the simulation. The charts were modified to be more easily readable. First, the displacement were averaged in order to remove small oscillations in successive iterations, therefore each point in the chart actually

represents an average of ten successive iterations. Second, the first five iterations were omitted. The displacements at the beginning of the simulation are very large, relative to the rest of the process, so incorporating them in the chart would reduce its readability. The aim of this test is to find out which layout converges fastest and therefore these several first iterations are not of interest anyway.

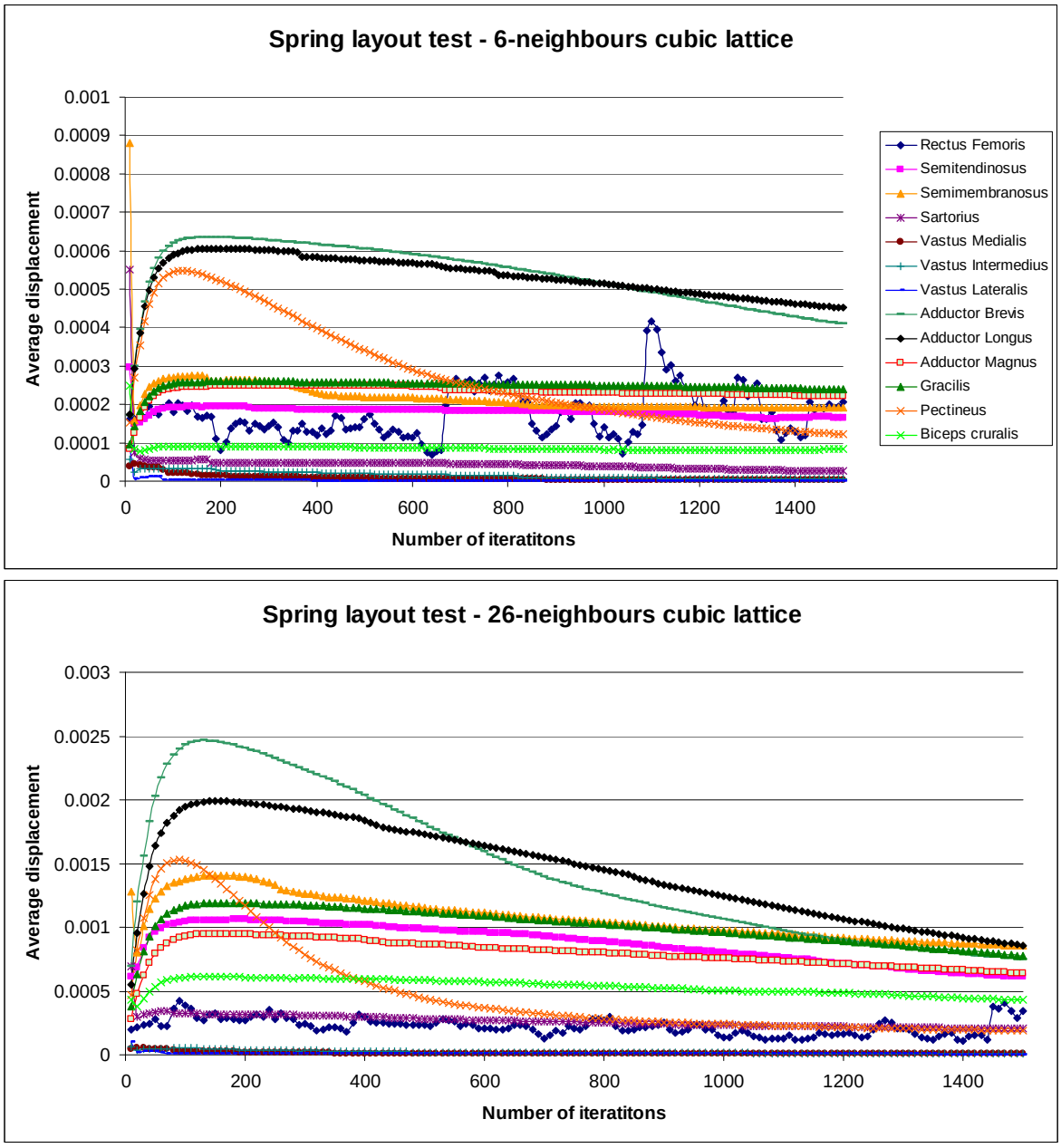

Figure 5.3a: Average displacement charts of the cubic lattice spring layouts.

The first noticeable fact that appears when comparing all four charts is that the absolute value of displacement is on average four times lower for the 6-neighbour cubic lattice than for the other layouts. However, the displacements only measure the changes between iterations, not how close the result actually is to the desired final position. For that purpose, the slope of the curve is more interesting. Figure 5.4 documents slopes for part of the curve (after being approximated by linear regression) for the Abductor Longus muscle. It proves what is visible from figures 5.3 and that is that the slope of the 6-cubic model is the

smallest – nine times smaller than for the 26-cubic model and five times smaller than for the other layouts. It could be said that this means nine times (or five) faster convergence of the system.

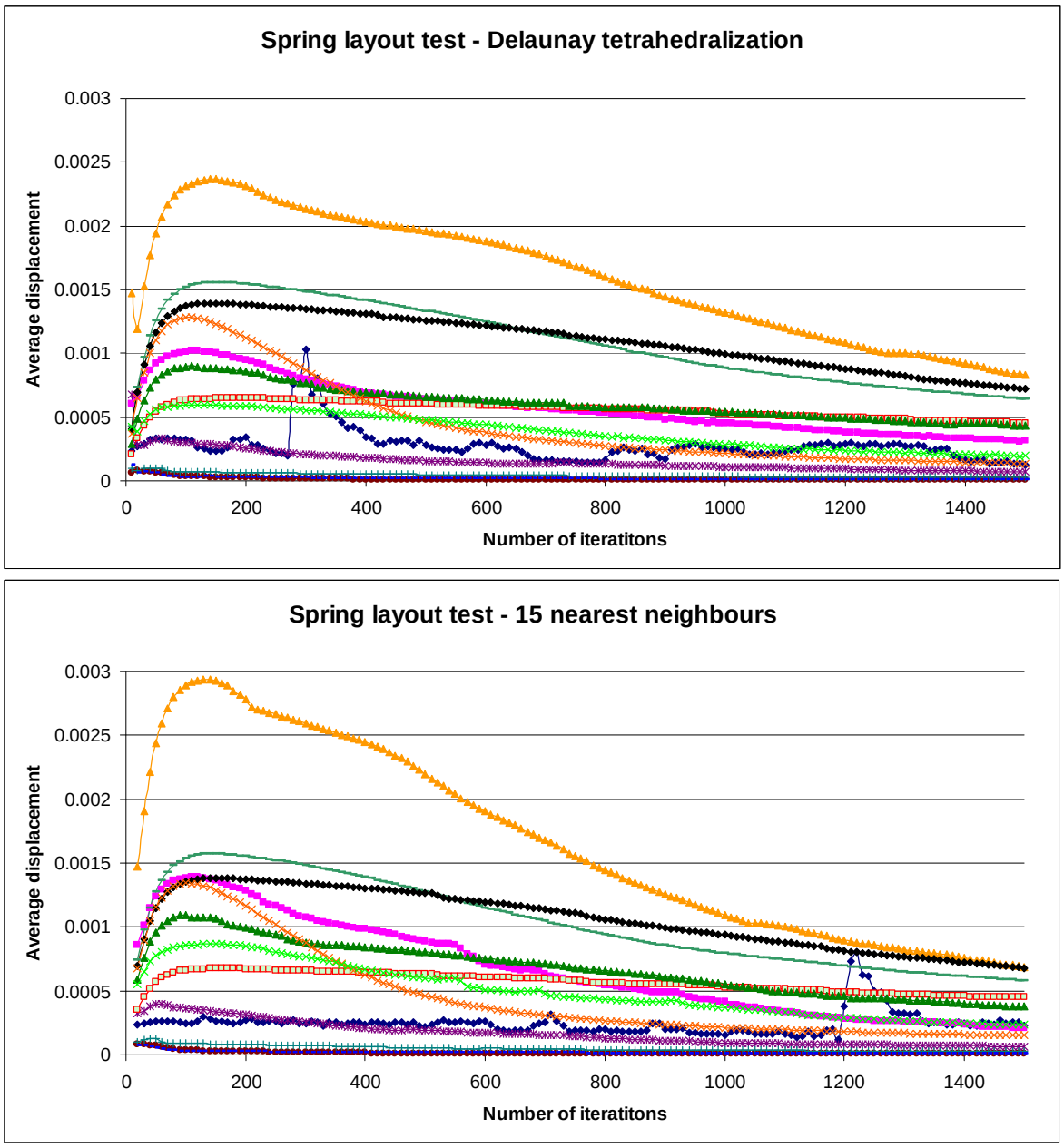

Figure 5.3b: Average displacement charts for the Delaunay tetrahedralization (upper) and 15 nearest neighbours spring layouts. The legend is the same as in figure 5.3a.

Figure 5.6 only further supports this conclusion. It shows the resulting shape of muscle fibres after the test. Most notable changes in shape are for the red and pink muscles (*Biceps cruralis* and *Semimembranosus*). Figure 5.7 shows these muscles deformed using the 26-model with more iterations and it is obvious that the shape to which the methods converge is such as achieved with the Delaunay and nearest neighbour layouts.

It can also be noted that the performance of the nearest neighbour and Delaunay layouts are very similar. This is actually not that surprising as most of the vertices required to

create a Delaunay tetrahedra connected to a given vertex will be the ones closest to it. Therefore, most of the spring in both layouts will be the same.

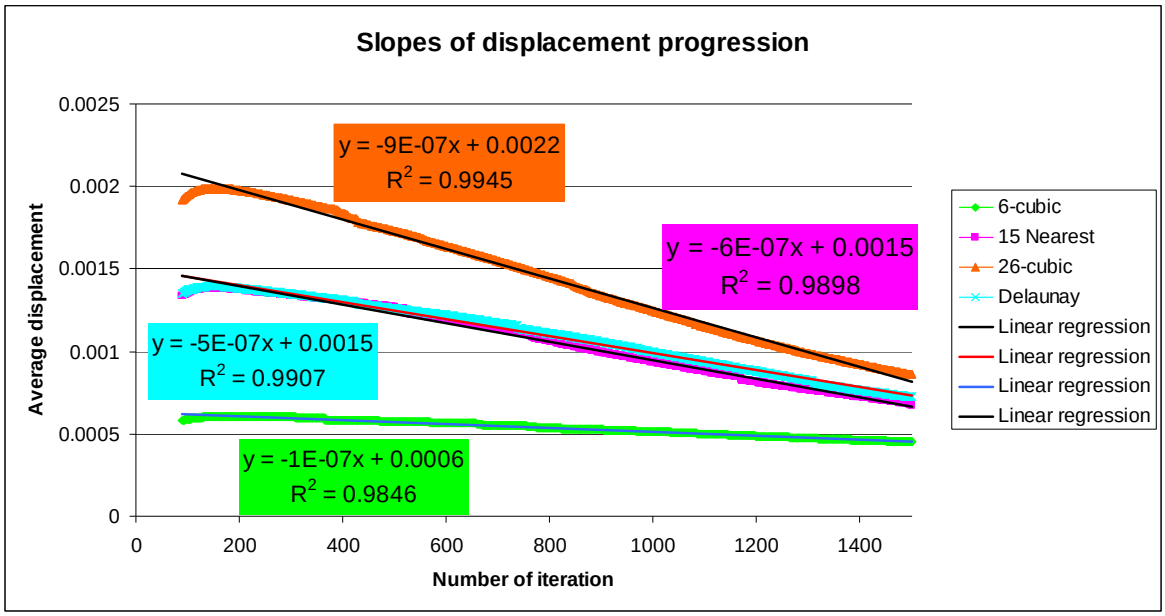

Figure 5.4: The progression of average displacements for all tested methods for the Abductor Longus muscle, approximated by linear regression. The regression formula for each method has the same background colour as the curve belonging to the given method and contains the reliability R^2 *of the regression.*

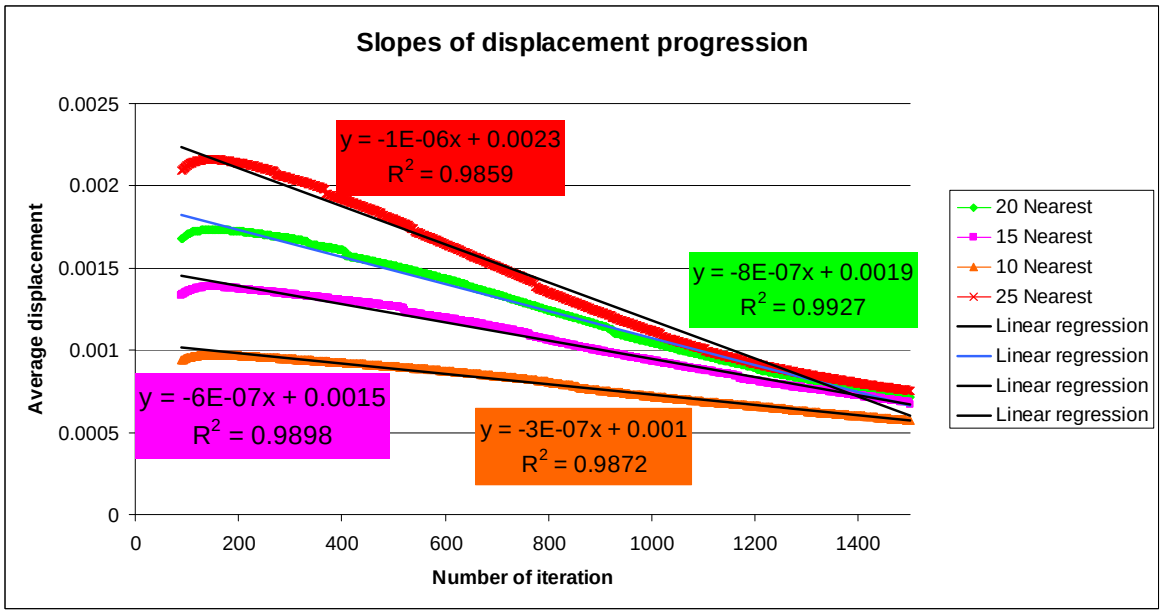

Figure 5.5: The progression of average displacements for the nearest neighbour method with various choices of the number of neighbours, approximated by linear regression. The subject muscle is the Abductor Longus. The regression formula for each choice of the number of neighbours (10, 15, 20, 25) has the same background colour as the curve belonging to the given number and contains the reliability R^2 *of the regression.*

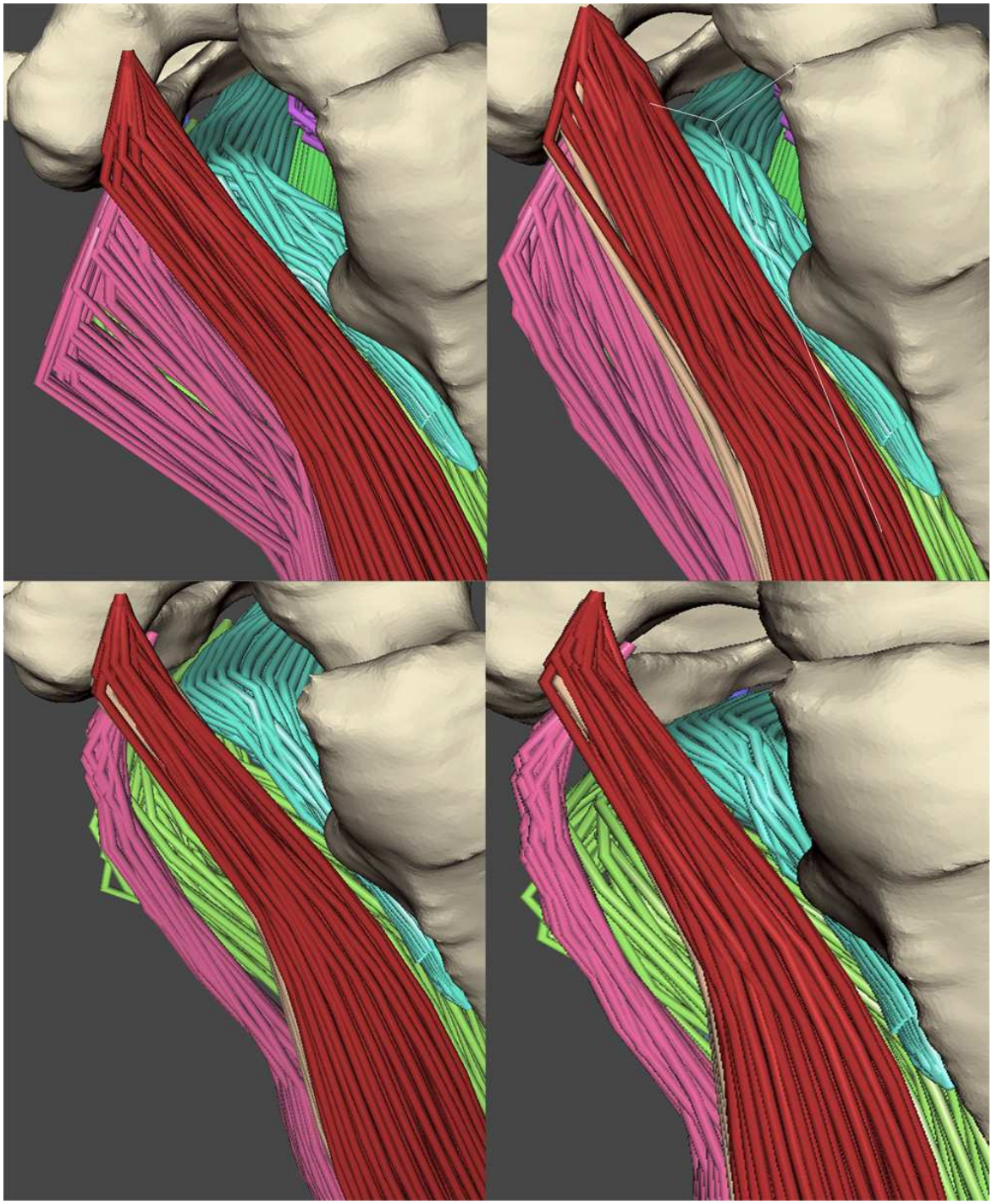

Figure 5.6: Visual comparison of the resulting deformation after 1500 iterations of muscle fibres for various layouts: upper left is 6-neighbours cubic, upper right 26-neighbour cubic, lower left Delaunay tetrahedralization and lower right 15 nearest neighbours.

The 6-neighbours cubic model turns out to be the least suitable. Although its time per iteration is the lowest, much more iterations are needed in order to achieve the same shape as when using the other layouts. The charts in figures 5.3a,b and 5.4 show that the particles of the 26-neighbour cubic model tend to stabilize more quickly, however, by visual comparison provided in figure 5.6, the other two layouts seem to converge to the correct shape more quickly. Also, they are slightly faster per iteration than the 26-neighbours model. The N-nearest neighbour is therefore suggested as the best of the available choices.

While it can achieve more or less the same results as the Delaunay tetrahedralization layout, the ability to choose various settings of the number *N* of nearest neighbours makes it more adaptive to various inputs.

Figure 5.5 compares the slopes of various choices of the number *N* of nearest neighbour for the nearest neighbour layout. Choices of 10, 15, 20 and 25 neighbours were tested and it can be seen that the relation between the average displacement slopes and the number of neighbours per particle is linear. However, for the 25 neighbours, the slope of the actual curve of average displacement, not its regression, seems to be decreasing with increasing iterations while it is almost perfectly linear in the other cases (at least in case of this experiment). This means that the difference of convergence speed between 25 and e.g. 20 neighbours would not actually be linear. Therefore, the suggested choice of the number of neighbours is between 15 and 20, because the balance between the speed of convergence and number of springs (and therefore time and memory requirements) is best.

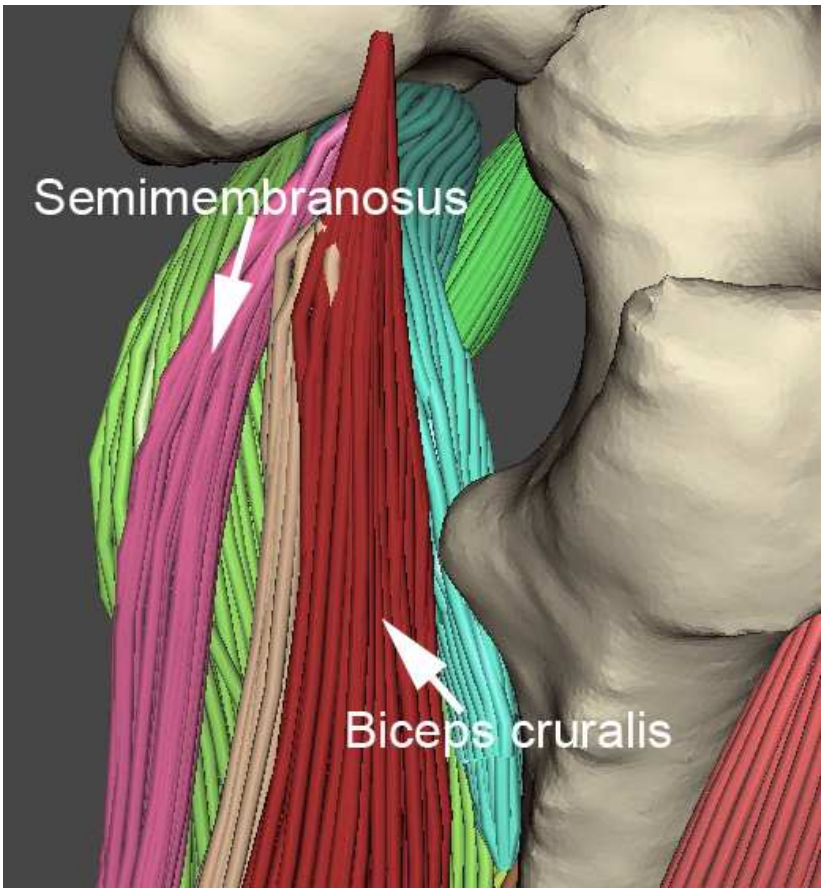

Figure 5.7: Thigh muscles deformed using the 26-neighbours cubic layout after 3000 iterations.

5.2. Collision detection mechanism settings

There is not much that can be modified on the CD mechanism. What can be altered in many different ways are the stopping conditions of BVH traversal, i.e. the minimum of primitives that may be in a leaf node and the maximal number of subdivision of the bounding volume. However, the impact of doing so it has on the overall performance is little to none. Section 3.1 suggested that the rebuilding of the BVH might not be necessary

and could actually improve the speed of the CD. Therefore, it the performance of the CD mechanism with and without rebuilding was tested.

The tested simulation step consisted of 70 iterations, i.e. 70 collision detections. The tested data were all thirteen available muscles on thigh and a total of eleven bones, located on both legs and in the pelvis area. Therefore, half of the CD against bones ended in the first step of the BVH traversal, as the bounding boxes of the muscles were not intersecting with bounding boxes of the left leg bones. On the other hand, each muscle collided with almost every other muscle involved in the simulation. Two positions (times) of the walking model were simulated, the initial position (*t = 0*) and the middle position (*t = 0.78*). Table 5.1 contains resulting times for the experiment. The time was measured separately for CD in between muscles, muscles and bones, the update phase and the collision response. The number of boundary particles (i.e. those that are used for CD) of the involved muscles were 1094 (eight muscles), 1182 (four muscles) and 1220 (one muscle). The numbers of spheres for the right leg bones and pelivs were 2987 for the right femur (thigh bone), 335 for the patella (knee cap), 1586 for the right fibula (calf bone), 2422 for the right tibia (shank bone) and 3211 and 3041 for the iliac and sacrum (pelvis) respectively.

	Update [s]	**Muscle CD [s]**	**Bone CD [s]**	**Response [s]**	**Total [s]**
Rebuilds, t = 0	0.078	10.966	10.847	0	21.891
Rebuilds, t = 0.78	0.016	10.404	10.705	0.015	21.140
No rebuilds, t = 0	0.016	10.445	10.480	0	20.941
No rebuilds, t = 0.78	0.016	10.907	9.407	0.015	20.345

Table 5.1: Computational times for the collision detection experiment. The times were achieved over 70 iterations. First two lines contain the results for the CD mechanism with rebuilding during the update phase; the remaining two are without rebuilding.

Although the CD without rebuilds is faster, the improvement is only about 5%, which contradicts the Van Den Bergen [25] statement about ten times faster computation when rebuilding is disallowed. However, it is important to realize that the approach that was implemented actually contains a test of whether the rebuild is needed or not. Therefore, the time saved by denying the rebuilds is not actually the time the method with rebuilds spends by rebuilding, but only the time it spends by checking if a rebuild is necessary (and finding out that it is not).

To sum up, employing the “no rebuilds” approach in the discussed application will make it slightly faster for the expected data sets (like the one used for testing). However, if the more robust approach which uses rebuilds will be used instead, no big harm will be done and the application will be ready for complex input. While this advantage is irrelevant now, it might become relevant should the application allow continuous animation of the scene (as discussed in section 4.1) later.

5.3. Setting the time step

The convergence of the mass-spring system is highly dependant on the magnitude of the time step *dt*. It should be as large as possible, but if it is too big, the system can diverge, as the velocities accumulated in the particles in each step will be too large. However, this is actually of no concern in the case of musculoskeletal model, as the movement of the particles is very limited due to collisions with all the neighbouring objects. But this holds

true only if the collisions can be correctly detected. This means ensuring that the tunnelling effect, discussed in chapter 3, will not appear.

Imagine a state of the particle system after successful collision detection, i.e. no particles are intersecting. In the worst case, a pair of particles will be touching and moving directly against each other in the next step. To ensure, that the tunnelling effect will not occur, the maximal distance they should be allowed to cover before the CD is invoked is equal to their radiuses (minus some infinitesimal ε). That way, they will almost coincide, but the CD will still be able to push them away from each other in the correct direction. But if they cover larger distance, the CD might not detect the collision at all or it will detect it, but resolve it by pushing the particles in a wrong direction. Therefore, the time step should be set in such a way that the boundary particles (that take part in the CD process) will not be able to travel larger distance than their radius.

Equation (2.14) presented a formula by which the new position of a particle $x_i(t+dt)$ is computed. The distance between the new $(t+dt)$ and old position (t) must be lesser or equal to the radius of the given particle, i.e. $Length\ (x_i(t+dt) - x_i(t)) <= radius_i$. Equation (5.1) shows the equation for determining the size of the time step dt derived from the equation (2.14) and the aforementioned relation between the radius and the covered distance. The external forces F^e (gravity etc.) are omitted from the equation, as they are not employed in the solution. The length of the spring forces were shortened to *LenF*, the length of the vector $x_i(t) - x_i(t-dt)$ was shortened to *LenX*.

$$
\begin{aligned}
&Length(x_i(t+dt) - x_i(t)) <= radius_i \\
&Length\left(\frac{-\sum_{\forall j \in N_i} F_{ij} - c\dfrac{x_i(t) - x_i(t-dt)}{dt}}{m_i} dt^2 \right) + \\
&\quad + Length(2x_i(t) - x_i(t-dt) - x_i(t)) <= radius_i \\
&LenF \cdot dt^2 - c \cdot LenX \cdot dt + m_i(LenX - radius_i) <= 0
\end{aligned}
\tag{5.1}
$$

Using the derived equation, the maximal possible *dt* is found for each particle (by solving the equation (5.1) as equality), and the lowest from among them is chosen as the maximal possible *dt* for the given mass-spring system. This is done for each soft-body in the scene and the lowest *dt* is set as the time step used for the upcoming MSS iteration to ensure that an equal amount of time is integrated for each object. This approach divides the MSS iteration into two phases – in the first one, the acting spring forces are computed and the maximal possible *dt* is found. Then the minimal *dt* among all the objects is set as the time step for each object and the second phase of MSS iterations is invoked – the update of the positions. After that, CD and CR are made and that concludes one simulation iteration.

Although the computation of the *dt* does take some extra computational time, it is not the main problem with this approach. It is clear that if there is a fast moving particle with a very small radius, it will force the time step to be very small, even though it is very unlikely that the small particle will actually collide with anything. This makes the approach very sensitive to the input, mainly to how the particle radiuses for muscles are set. Section 4.3 described the heuristic used for these settings. The distances between the vertices, to

which a given particle is the closest and the particle itself are used to set the radius. This works well when there are a lot of vertices “belonging” to the particle. But in some cases, the particle might be very close to each other, resulting in a very small radius, because it is set in such a way that the particles do not intersect (see section 4.3).

Another problem is caused due to the way the particles are transformed before the soft-body simulation begins. Because each particle is transformed according to the closest bone, the particles usually get separated by a large distance after the transformation is made. Figure 5.8 documents this on several muscles on the thigh. It is clearly visible that the particles in the upper parts of the muscles were bound to a different bone (pelvis) and therefore transformed by different transformations than the particles in the lower parts. This results in large tension (forces) in the springs and therefore a relatively fast movement. But if this movement is to be limited by the radius of the particle, the time step must be very small.

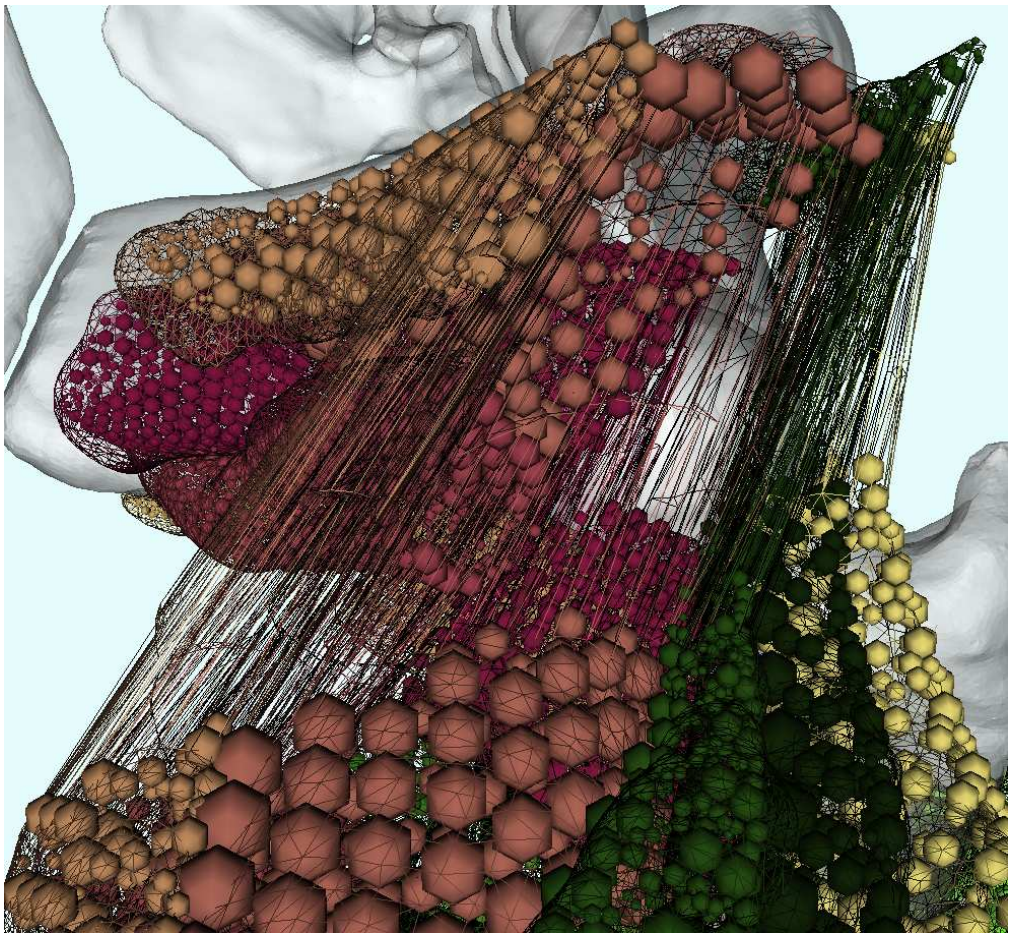

Figure 5.8: Colour-coded particles of several muscles on the thigh with a wireframe visualization of the surface after rigid transformations, before soft-body simulation.

The consequence of these two reasons is that the time step becomes unnecessarily small – many of the particles responsible for the slow down will not collide anyway so a larger time step would not damage the simulation. But there is no way to tell beforehand which particles shall be allowed to cover longer distance than their radius. Sadly, the solution of this problem has not been found yet. Nevertheless, this approach guarantees correct collision resolution, therefore it is the proposed approach.

The only alternative is to experimentally determine a “safe” constant time step for the given data set, i.e. such that does not produce any visible anomalies, and use it, which is certainly not an ideal approach as well. But, if time is critical, it might be the only option to generate a plausible output, even though it might contain occasional intersections.

5.4. Number of iterations

The iteration count of each simulation step should obviously be as small as possible in order to achieve fast performance. On the other hand, the MSS needs should ideally converge to a stable state where no particles move and many iterations are needed for that. The aim of the following experiment is to find a suitable compromise between these two, i.e. the lowest possible number of iteration during which the MSS produces acceptable results. The subjects of the test were again the thirteen muscles on the thigh, transformed into the initial position of the moving data set.

The metric used in experiment 5.1, i.e. the average displacement of particles, is not very suitable for finding out how close the current state of the mass-spring system is to the final state, because the truly final positions are not known. However, what is known is the final length of the springs – it should be the same as the initial length. Therefore the chosen metric is the sum of differences between current spring lengths in a given iteration and the initial rest lengths.

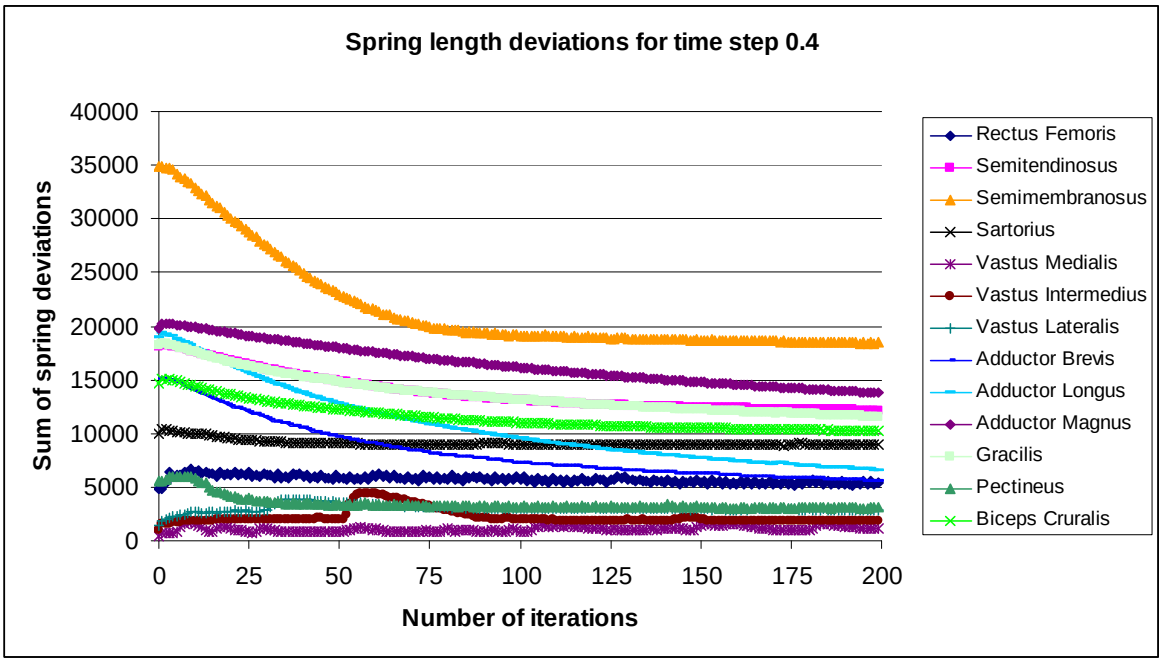

Figure 5.9: Spring deviations for thigh muscles with dt set to 0.4 over 200 iterations.

Figure 5.9 shows the progress of deviations for each of the tested muscles from the beginning to the two hundredth iteration. The time step was set as fixed to the value of 0.4. It can be observed on the Semimembranosus muscle that after approximately seventy five iterations, the progress slows done rapidly. This means, that the changes in the shape of the muscle are not very noticeable, therefore the “value” each subsequent iteration gives to the output is lower. The target number of iterations should be set so that the ratio between value (contribution of the iteration to the final shape) and cost (time) becomes too low after that many iterations. What should be considered as too low is a matter of the sought

precision. By simply looking at the chart, seventy five iterations would seem like a good choice as far as the Semimembranosus muscle is concerned, as the added value of each iteration after that is much lower than it was in the previous iterations.

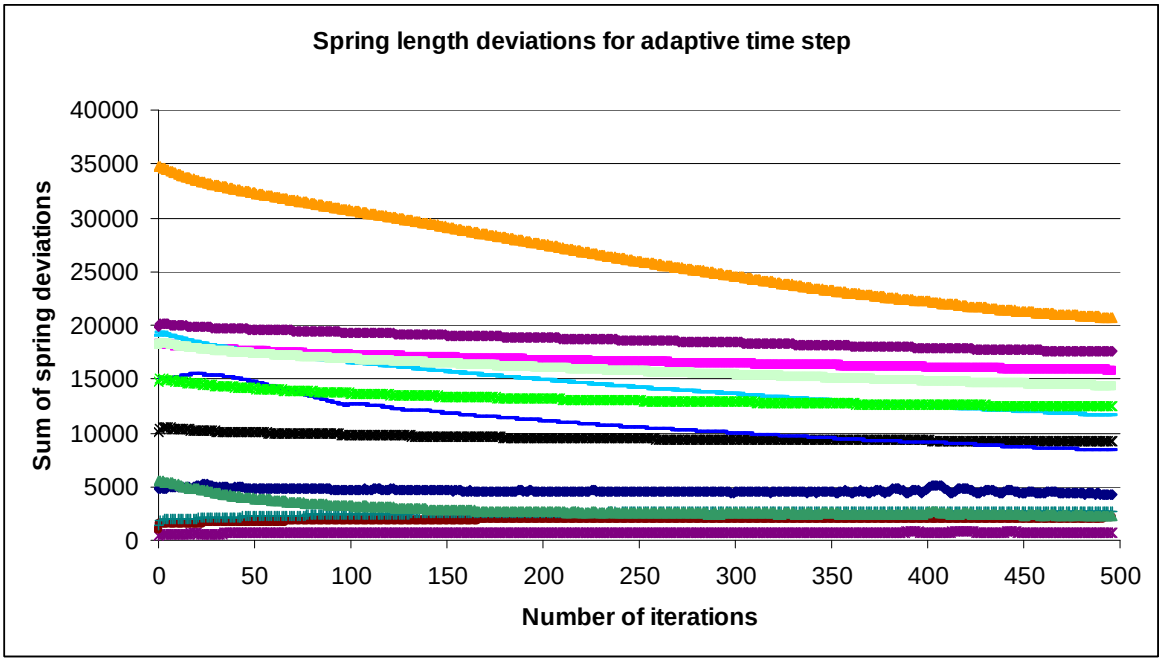

Figure 5.10: Spring deviations for thigh muscles when adaptive step is used for 500 iterations. The legend (muscle colours) is the same as in figure 5.9.

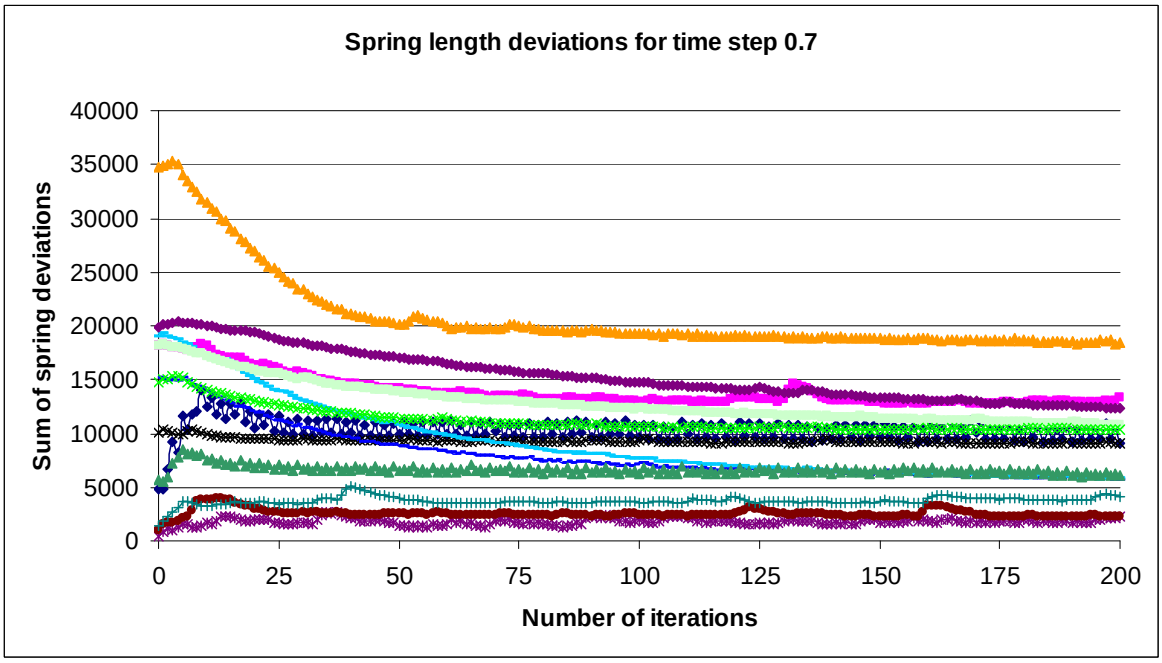

Figure 5.11: Spring deviations for thigh muscles when dt 0.7 is used for 200 iterations. The legend (muscle colours) is the same as in figure 5.9.

When other muscles are concerned, the decision is less easy. For some muscles, such as the Semitendinosus or Gracilis, the progress tend to the hyperbolic shape as in the case of

Semimembranosus and the threshold of seventy five iterations would also look reasonable, for muscles such as the Abductor Magnus, it is practically impossible to detect any point where the progress becomes lower than it was before, as it is almost perfectly linear.

More problems arise when a lower time step is chosen. Figure 5.10 shows the resulting spring deviations when the adaptive time step is used. The minimal time step was 0.04 – the adaptive time step was not allowed to be lower. In most of the iterations, this minimal step was used, because the adaptive time step (see section 5.3) was even lower (an average of 0.046782 was achieved over 1000 iterations). It can be seen in the chart that the progress is very slow since the beginning and it does not improve in time, making it hard to decide when to stop the simulation.

If a higher time step is used, the progress is indeed faster as figure 5.11 documents (time step 0.7). If figures 5.9 and 5.11 are compared, it can be seen that the spring deviations achieved in the first case after seventy five iterations are achieved in the second case in less than fifty. However, the sudden abrupt changes of the deviations in numerous cases in figure 5.11 are foreboding errors in the simulation, probably due to faulty collision detection or divergence of the mass-spring system. Figure 5.12 confirms this visually as the endings of the muscle fibres are flawed.

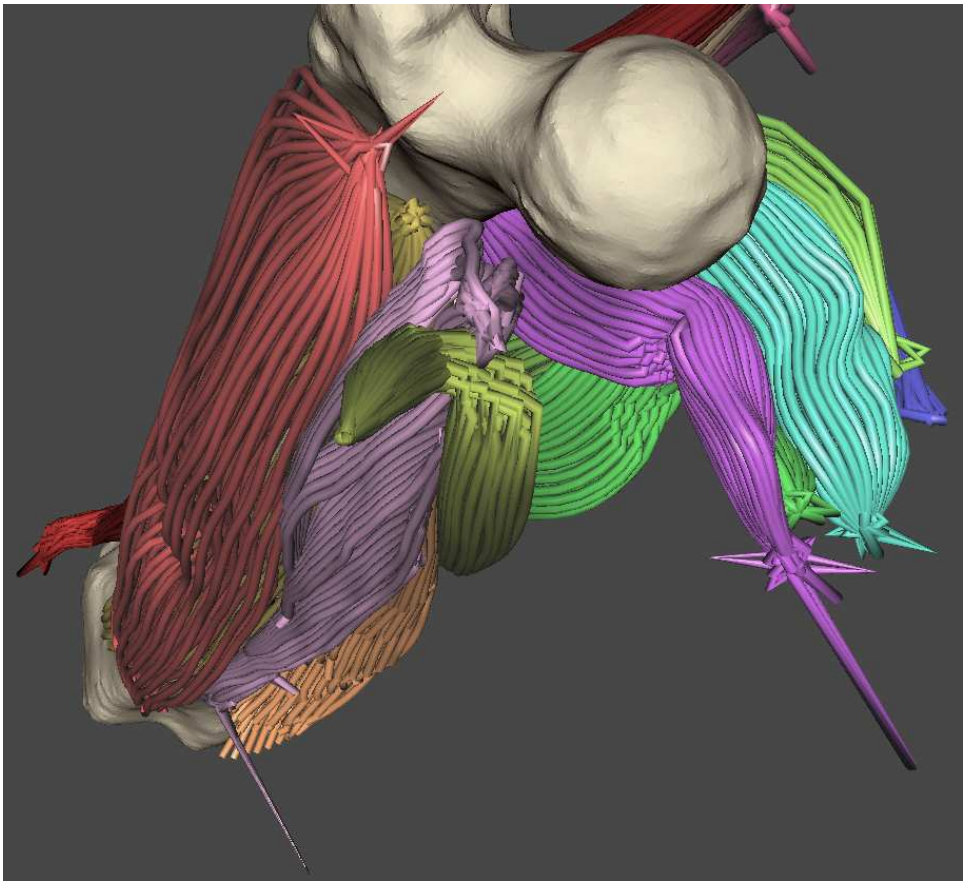

Figure 5.12: Thigh muscles and bone after 1000 iterations with fixed time step dt = 0.7.

To sum up, there is no easy way to tell how many iterations should be used for the. The ratio between the spring deviation and total rest length of the springs could be used to decide when to stop the simulation (e.g. when the ratio is less than 0.1, 0.01 etc.), but the

setting of the threshold value would also have to take into account the size of the time step. The situation would get easier if a requirement of maximal computation time was assigned, the system would then simply run for as long as it would be allowed.

5.5. Surface deformation quality

The LHPBuilder application currently contains another deformation method [10], called "PK method", which handles multiple objects in the scene, i.e. accounts for collisions. This method is compared with the approach proposed in this thesis in figure 5.14. It is immediately visible that the mass-spring method performs much better in terms of visual plausibility than the PK method, which tends to "carve" the bone into the muscle. Figure 5.13 shows the same scenario from different point of view for the PK method, which makes the problem even more apparent.

The scenario depicted in figures 5.13 and 5.14 shows a deformation as a result of contact of two objects and it seems that the designed method handles it well. The situation is different for the deformations that occur due to the transformations made at the beginning of the simulation. This is a problem for the simulation as whole (already discussed in section 5.3), because it takes many iterations of the MSS to converge from this state to some "realistic" state where the two parts join again. This results in large artefacts between these two sections of the muscles. If enough iterations and large enough time step is used, the particles will eventually join and the artefacts will not be noticeable on the fibres. Unfortunately, as figure 5.15 shows, this is not true for the surface, which still suffers from the artefacts.

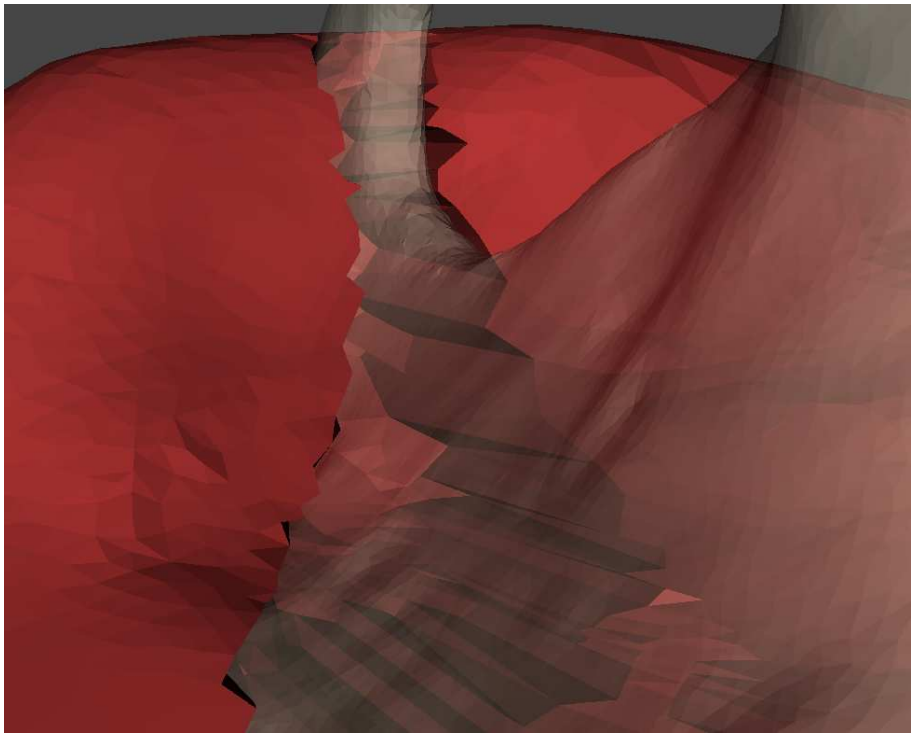

Figure 5.13: Deformation of the surface of the Gluteus maximus muscle as a result of contact with pelvis. The PK method was used. The bone is visualized as a white opaque surface, while the muscle is the red solid surface.

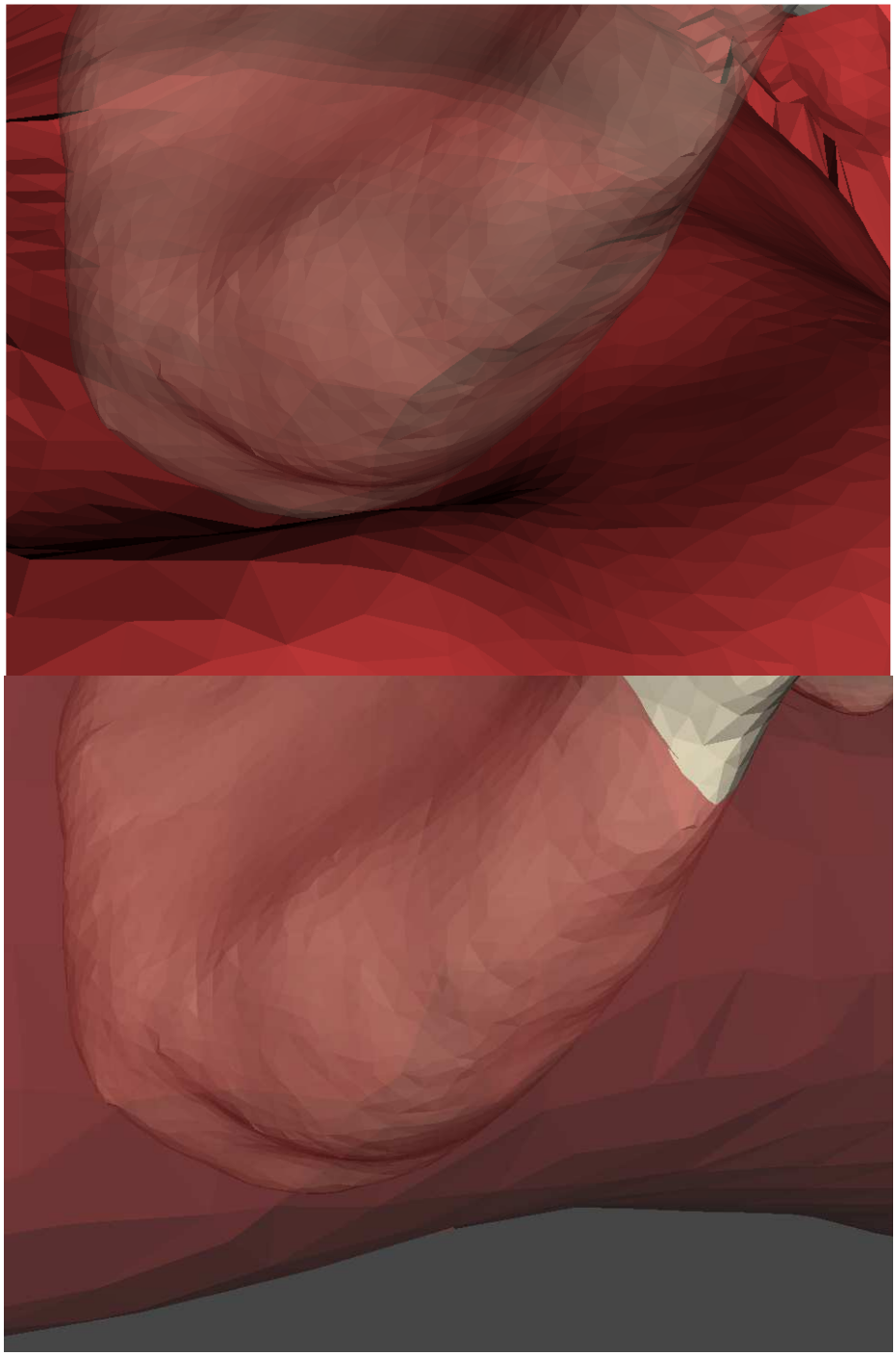

Figure 5.14: Deformation of the surface of the Gluteus maximus muscle as a result of contact with pelvis. The proposed mass-spring method was used for the upper image; the PK method was used for the lower image. The muscle is visualized as a red surface and bone as white. The bone is slightly opaque on the upper image to make the muscle visible, while on the other hand the muscle is opaque in the lower in order to make the bone visible. The camera angle and point of view is the same for both images.

The figure 5.16 shows the same scenario with two different setting of the number of nearest particles associated with each vertex. One uses three nearest particles, the other eight. The difference, however, is effectively none. Also, no dramatic improvement was obtained when more particles were used for the muscles.

This makes the proposed method rather unattractive to use. Metrics such as volume preservation were not even measured, as it is clear that the method would not do very well. Please note that this does not make the whole mass-spring method unsuitable – the deformation of muscle fibres, which is the most important output, is still usable. However, different method for the surface deformation has to be found.

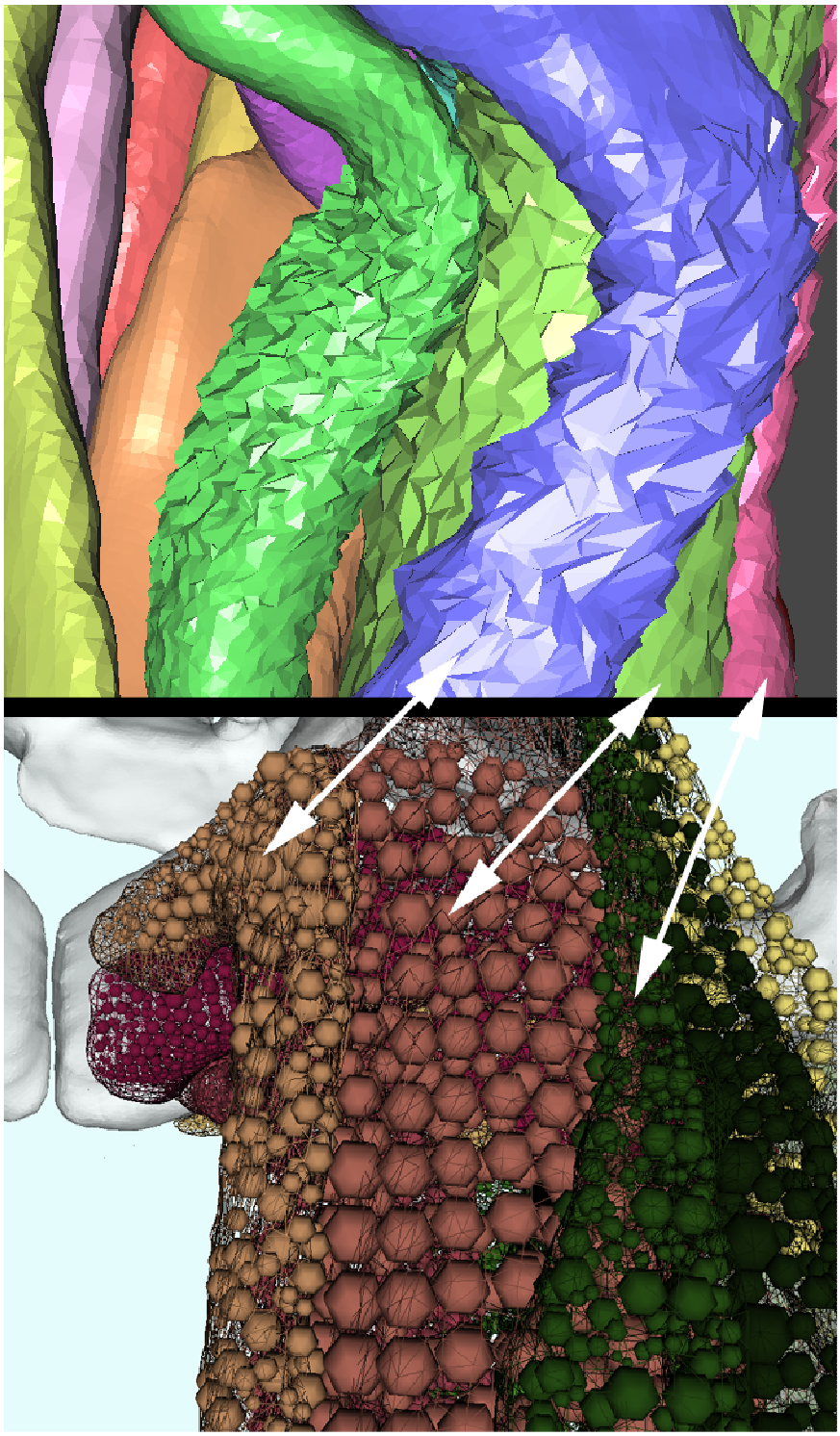

Figure 5.15: The deformed surface (upper) and the particles (lower) of several muscles after 1000 iterations with large time step were applied. The images were captured from different views - corresponding muscles are marked by arrows.

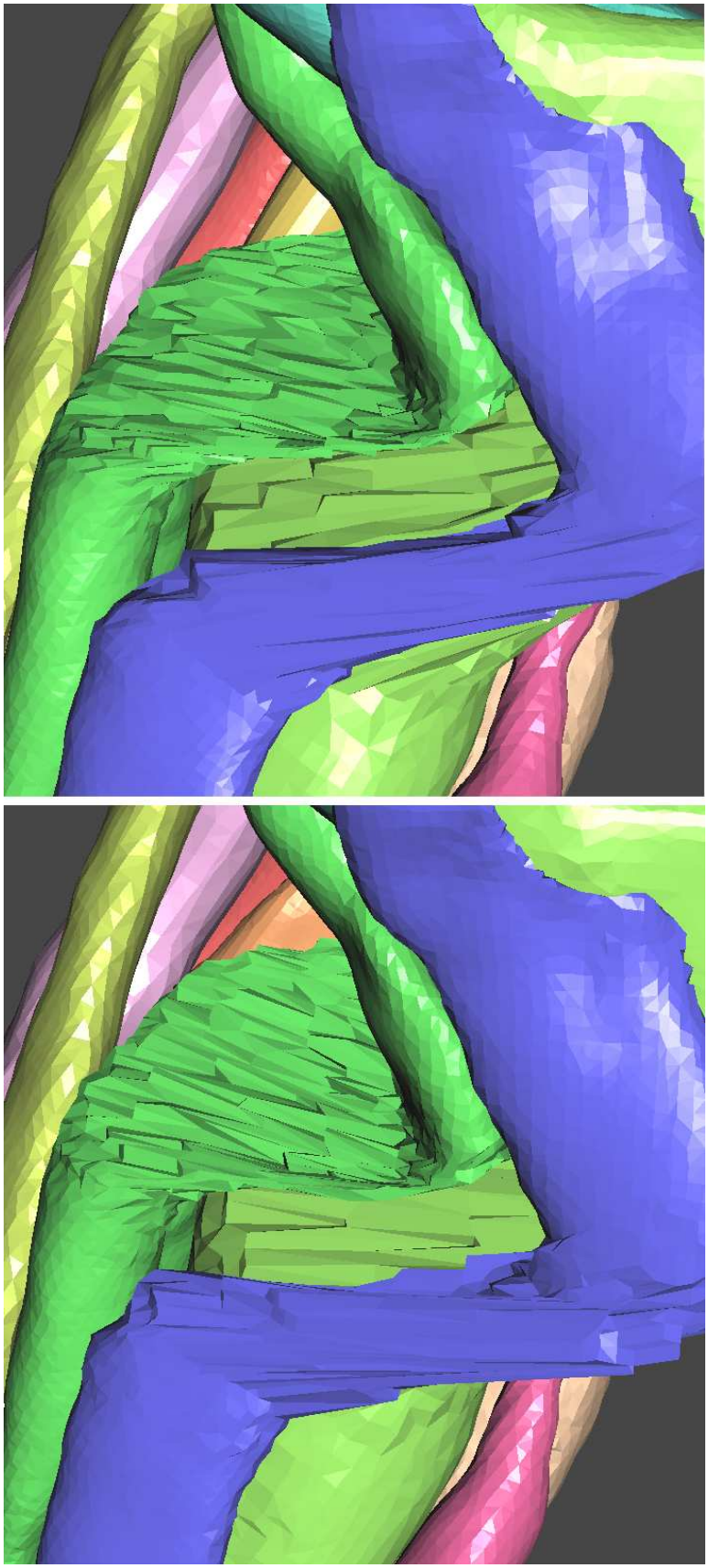

Figure 5.16: The deformation of several muscles on the thigh. Eight closest particles per vertex were used on the right image, three on the left. The particle positions for this scenario can be seen in figure 5.8.

5.6. Overall time performance

The muscles on the thigh were used to test the overall speed of the implemented solution. The test measured the time of the MSS processing and the CD consumed during a one hundred iteration long simulation. The spring layout chosen for the experiment was the nearest neighbour, using eighteen neighbours per particle.

The main factor that influences the computational time of one simulation iteration is the number of particles in muscles. The number of particles in bones also has impact, but it cannot be modified by the user, therefore it was set to the same number in all the following tests. To change the number of particles in muscles, four different settings of the fibre resolution, i.e. the number of particles per fibre, were tested – 20, 40, 60 and 80. The number of fibres was set to 64 in all cases. Despite that, the particle count does not necessarily have to be 64 times the resolution for each muscle due to some implementation specifics. Table 5.2 shows the number of particles of given muscles. The number of boundary particles and total number of particles is given separately, because only the boundary particles participate in the CD. The numbers of spheres for the most used bones were the same as in the experiment 5.2: 2987 for the right femur (thigh bone), 335 for the patella (knee cap), 1586 for the right fibula (calf bone) and 2422 for the right tibia (shank bone) and 3121 for the pelvis.

	Boundary particles				**Total particles**			
Resolution / Muscle	**20**	**40**	**60**	**80**	**20**	**40**	**60**	**80**
Rectus Femoris	622	1182	1765	2325	1306	2586	3889	5169
Semitendinosus	534	1094	1654	2214	1218	2498	3778	5058
Semimembranosus	534	1094	1654	2214	1218	2498	3778	5058
Sartorius	534	1094	1654	2214	1218	2498	3778	5058
Vastus Medialis	660	1220	1780	2340	1344	2624	3904	5184
Vastus Intermedius	534	1094	1654	2214	1218	2498	3778	5058
Vastus Lateralis	534	1094	1654	2214	1218	2498	3778	5058
Abductus Brevis	558	1118	1678	2340	1242	2522	3802	5082
Abductus Longus	558	1118	1678	2340	1242	2522	3802	5082
Abductus Magnus	558	1118	1678	2340	1242	2522	3802	5082
Gracilis	534	1094	1654	2214	1218	2498	3778	5058
Pectineus	534	1094	1654	2214	1218	2498	3778	5058
Biceps Cruralis	534	1094	1654	2214	1218	2498	3778	5058
TOTAL	7228	14508	21811	29397	16120	32760	49423	66063

Table 5.2: Particle counts for thigh muscles in relation to the resolution of the fibres.

The performance of the CD is not dependant only on the number of particles, but also on the position of the objects. The results of the test should be interpreted carefully – the time spent by collision handling for the same number of particles may vary significantly in different situation. For this reason, five different positions on the timeline of the walking data set were tested and the times for the CD were averaged. On the other hand, the computational time of the MSS should depend solely on the total number of particles.

Table 5.3 contains the times for collision detection for the five individual positions, denoted by their time (the simulation of walking spans between times $t = 0$ and $t = 1.56$). Notice that the differences between individual positions are indeed significant and even more so with increasing number of particles. An average for each fibre resolution is given. The CD without rebuilding was used for the measuring.

Resolution / Position	20 [s]	40 [s]	60 [s]	80 [s]
t = 0	21.159	31.596	42.453	53.270
t = 0.4	16.170	27.769	37.393	42.955
t = 0.8	19.873	30.498	47.251	50.688
t = 1.2	16.311	27.763	36.550	39.639
t = 1.56	18.932	31.593	43.688	64.327
AVERAGE	18.489	29.844	41.467	50.176

Table 5.3: Computational times of collision detection in five different positions of the walking data set.

Figure 5.17 shows the resulting average times of CD in relation to the total number of boundary particles. Surprisingly, there seems to be a strong correlation between these quantities. It could be concluded that in average case, the time needed for CD is linearly dependant on the number of particles. A more detailed test, preferably with several different data sets, would be in order if one wanted to prove or disprove this statement.

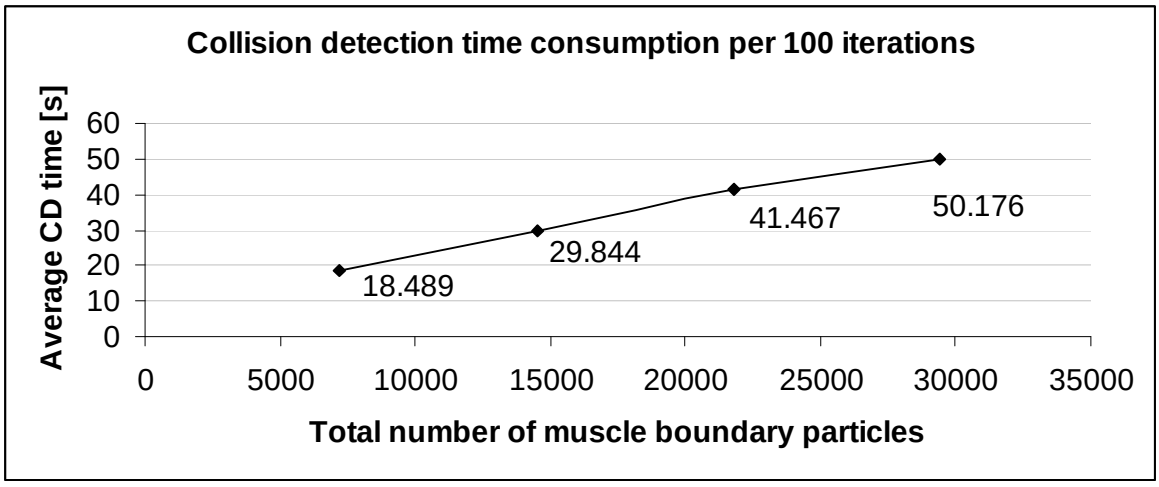

Figure 5.17: Average computational time of collision detection as a function of the number of boundary particles. The depicted time is the time accumulated in 100 iterations.

Figure 5.18 shows the time consumption of the MSS simulation as a function of the total number of particles. The position $t = 0$ was used to create this chart, but the times are almost the same for all the positions as expected. Again, a linear dependency on the number of particles can be observed.

It is immediately obvious that the CD mechanism is the bottleneck of the whole method and therefore should be the primary target for future optimization. If the case of resolution 40 is considered as a reasonable amount of particles to represent the muscle, a total time of 35.6 seconds per one hundred iterations is obtained. This means approximately three

iterations per second. According to section 5.4, the one hundred iterations is actually more or less the minimum number needed to achieve a plausible deformation, therefore this method is far from being interactive. However, it manages to produce several frames of simulation in a matter of minutes, while FEM based methods require hours to do the same.

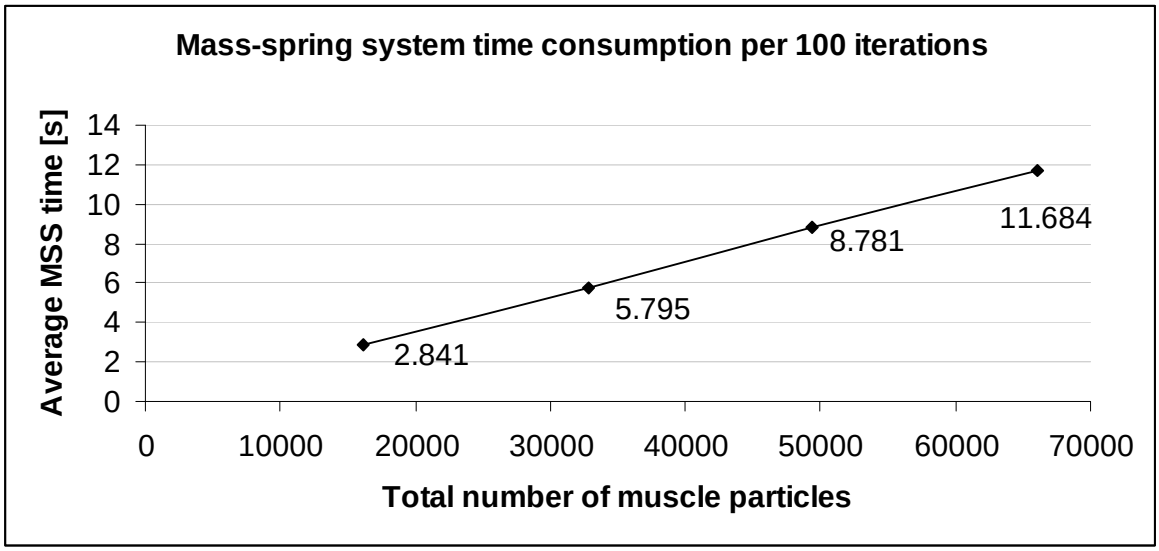

Figure 5.18: Computational time of mass-spring system simulation as a function of the number of particles. The depicted time is the time accumulated in 100 iterations

6. Conclusion

A mass-spring model designed for representing muscles in a musculoskeletal model was presented. The mass-points, or particles, for the model are obtained by sampling the fibres of the muscle, thus creating a volumetric representation of the muscle. Three different ways of connecting the particles by springs were tested – cubic lattice, Delaunay tetrahedralization and N-nearest neighbours. After evaluating the experiments, the N-nearest neighbours came up as the most suitable solution, allowing fast convergence of the model without using excessive number of springs.

A collision detection and response mechanism for the aforementioned model was designed, implemented and tested. The mechanism employs lazily updated bounding volume hierarchies to speed up the collision detection. In order to further enhance the speed of the collision handling, the particles of the mass-spring system are utilized to approximate the surface of each object. This approximation not only allows employing faster sphere vs. sphere tests instead of triangle vs. triangle, but mainly bypasses the need to propagate changes of the shape between the surface model and the mass-spring model in each iteration of the simulation. The drawback is that the collision response is not perfectly precise and therefore the surfaces of the objects might partly intersect. The mechanism allows collisions detection between two soft bodies as well as between a soft-body (muscle) and rigid-body (bone).

The proposed solution was implemented and tested using the LHPBuilder application, which allows the processing of data obtained from MRI scanning of a real human. Although a speed of up to five frames per second can be achieved for a semi-large data set (approximately ten bones and ten muscles), the fidelity of the result would be rather poor, although sufficient for quick visualization. In order to achieve higher quality results, less interactive speed of several to several tens of seconds is required. This is, however, still faster than the other method (PK method) capable of simulating multiple objects at once that is currently implemented in the LHPBuilder or any approach based on the Finite Element Method. The implementation allows the user to change the number of iterations used for the simulation, thus changing the ratio between speed and quality.

There are several known issues that should be fixed in the future. First one is the method used to update the surface model of the muscle, because the proposed method turned out to be flawed by some artefacts. Other issue is the speed of the proposed method. The implementation is currently very basic and effectively in a prototype state. During future refinement, both the mass-spring system solver and the collision detection mechanism should be parallelized. Especially in the case of the collision detection mechanism, which is the bottleneck of the method, the parallelization could yield large speed improvement. Although the each-with-each collision detection algorithm would in its basic form require the use of critical sections, a scheme that would efficiently process the collision detection between pairs of objects without conflicts can be designed as well. Moreover, the simplicity of the primitives used by the collision detection mechanism, i.e. spheres and axis aligned bounding boxes, makes the use of general purpose GPUs also a viable option.

The proposed solution offers a fast soft-body simulation, but it certainly cannot compete with the fidelity of the simulation of approaches based on the Finite Element Method. It is now up to medical experts, who will be testing the implemented software equipment in near future, to judge whether it is sufficiently reliable for their purposes.

References

[1] Bartels R., Beatty J., Barsky B.: *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, Los Altos, 1987.

[2] https://www.biomedtown.org/biomed_town/MAF/Reception

[3] Breen D., House D., Wozny M.: *Predicting the drape of woven cloth using interactive particles.* Proceedings of ACM SIGGRAPH Computer graphics and interactive techniques, Volume 28, pp. 365 – 372, 1994.

[4] Bridson R., Marino S., Fedkiw R.: *Simulation of clothing with folds and wrinkles.* Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 28 – 36, 2003.

[5] Debunne G., Desbrun M., Cani M.-P., Barr A. H.: *Dynamic real-time deformations using space & time adaptive sampling.* Proceedings of ACM SIGGRAPH Computer Graphics 2001, pp. 31 – 36, 2001.

[6] Ehmann S. A., Lin M. C.: *Accurate and Fast Proximity Queries Between Polyhedra Using Surface Decomposition.* Computer Graphics Forum, Volume 20 (3), pp. 500 – 510, 2001.

[7] Gibson S.F.F, Mitrich B.: *A Survey of Deformable Modeling in Computer Graphics*. Technical report TR-97-19, MERL – A Mitsubishi Electric Research Laboratory, 1997.

[8] Hutchinson D., Preston M., Hewitt T.: *Adaptive refinement for mass/spring simulations.* Proceedings of the Eurographics workshop on Computer animation and simulation, pp. 31 – 45, 1996.

[9] http://www.iofbonehealth.org/bonehealth/what-osteoporosis-1

[10] Kohout J., Kellnhofer P., Martelli S.: *Fast deformation for modelling of musculoskeletal system*. In Proceedings of Proceedings of the International Conference on Computer Graphics Theory and Applications: GRAPP, 2012.

[11] Landau L. D., Lifshitz E. M., Kosevich A. M., Pitaevskii L. P.: *Theory of Elasticity*. Elsevier, 1986.

[12] Larsson T., Akenine-Möller T.: *A dynamic bounding volume hierarchy for generalized collision detection.* Computer and Graphics, Volume 30 (3), pp. 451 – 460, 2006.

[13] Meier U., López O., Monserrat C., Juan M.C., Alcaňiz M.: *Real-time deformable models for surgery simulation: a survey*. Computer Methods and Programs in Biomedicine, Volume 77, pp. 183-197, 2005

[14] Mesit J., Guha R. K., Hastings E. J.: *Multi-level SB Collide: Collision and Self-.Collision in Soft Bodies.* Proceedings of the International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES'06), 2006.

[15] Mezger J., Kimmerle S., Etzmuß O.: *Hierarchical Techniques in Collision Detection for Cloth Animation*. Journal of WSCG, Volume 11 (1), pp. 322 – 329, 2003.

[16] Müller M., Dorsey J., McMillan L., Jagnow R., Cutler B.: *Stable real-time deformations.* Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 49 – 54, 2002.

[17] Nealen A., Müller M., Keiser R., Boxerman E., Carlson M.: *Physically based deformable models in Computer Graphics*. Computer Graphics Forum, Volume 25 (4), pp. 809 – 836, 2006.

[18] Sederberg T., Parry S.: *Free-form deformation of solid geometric models.* ACM SIGGRAPH Computer Graphics, Volume 20 (4), pp. 151-160, 1986.

[19] Tang Y. M., Zhou A.F., Hui K. C.: *Comparison between FEM and BEM for Real-time Simulation.* Computer aided Design & Applications, Volume 2, pp. 421 – 430, 2005.

[20] Terzopoulos D., Fleischer K.: *Deformable models*. The Visual Computer, Volume 4 (6), pp. 306 – 331, 1988.

[21] Terzopoulos D., Platt J., Barr A., Fleischer K.: *Elastically Deformable Models*. ACM SIGGRAPH Computer Graphics, Volume 21 (4), pp. 205-214, 1987.

[22] Teschner M., Heidelberger B., Müller M., Gross M.: *A versatile and robust model for geometrically complex deformable solids.* Proceedings of Computer Graphics International 2004, pp. 312 – 319, 2004.

[23] Teschner M., Heidelberger B., Müller M., Pomeranets D., Gross M.: *Optimized spatial hashing for collision detection of deformable objects.* Proceedings of Vision, Modeling, Visualization VMV'03, pp. 47 – 54, 2003.

[24] Teschner M., Kimmerle S., Heidelberger B., Zachmann G., Raghupathi L., Fuhrmann A., Cani M.-P., Faure F., Magnenat-Thalmann N., Strasser W., Volino P.: *Collision Detection for Deformable Objects.* Computer Graphics Forum, Volume 24 (1), pp. 61 – 81, 2005.

[25] Van Den Bergen G.: *Efficient Collision Detection of Complex Deformable Models using AABB Trees.* Journal of Graphic Tools, Volume 2 (4), pp- 1 – 14, 1997.

[26] http://www.vphop.eu

[27] http://www.vtk.org/

[28] http://en.wikipedia.org/wiki/Bounding_volume

[29] Zelený I.: *Vzájemná transformace 3D objektů reprezentovaných trojúhelníkovým povrchem.* Diploma thesis, University of West Bohemia, Faculty of applied sciences, 2011. (Available only in Czech language).

Appendix

- User manual
- Programmer’s manual
- Overview of appended material

User manual

This section will provide brief information on how to use the LHPBuilder demo that is provided with the thesis. To install the demo, simply run the LHPBuilderDemoInstall.exe provided on the DVD and follow the install wizard dialogue. Please not that it is a MS Windows application. After it is installed, simply run it from the installation directory.

The application loads the demonstration testing data (the same that were used for experiments in chapter 5) automatically. On the right side, there is a hierarchical tree with the data (area 1 in figure A.1). There are two "Atlases" in the tree. The first one represents the rest pose, while the second ("Atlas fused by walking") represents the walking data.

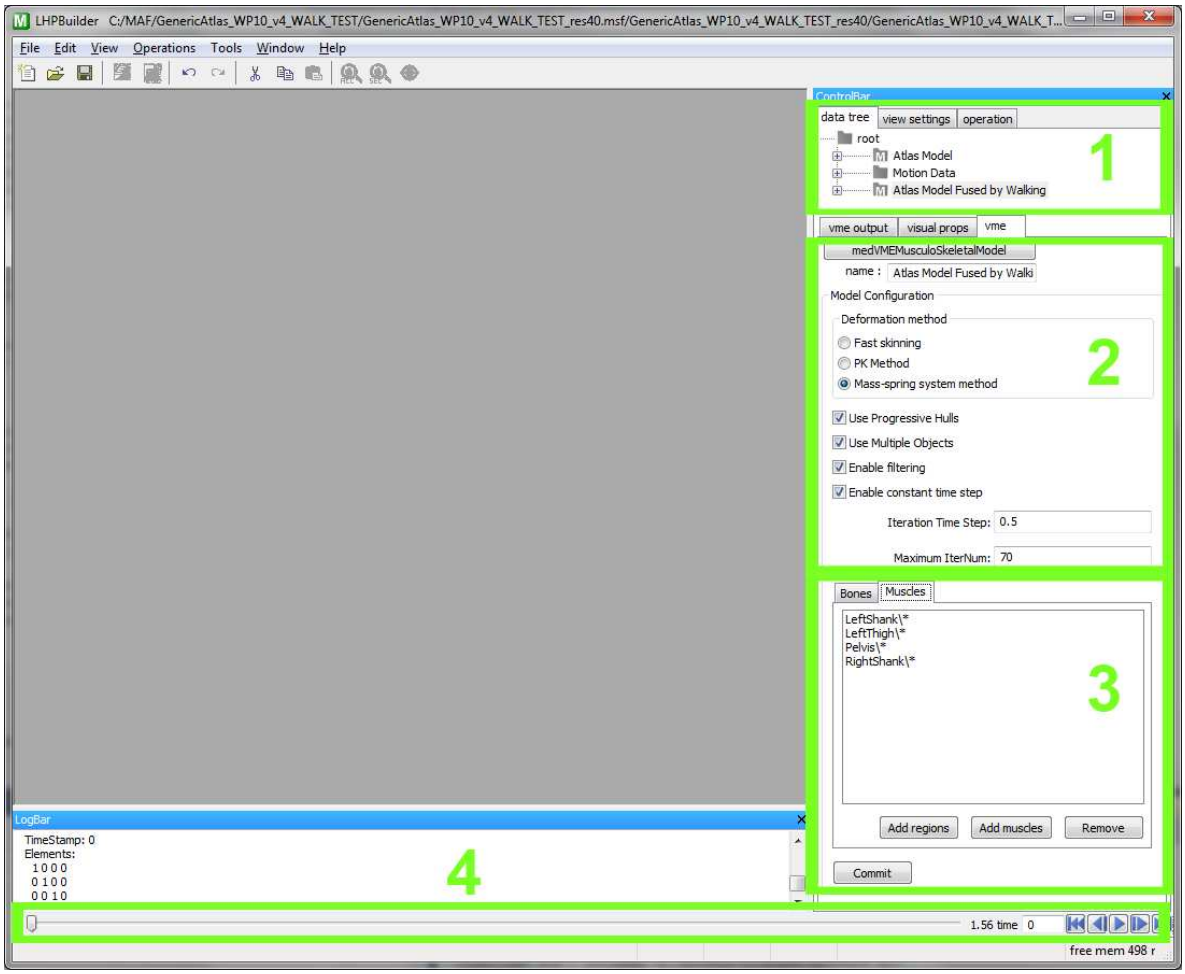

Figure A.1: The interface of the LHPBuilder application after the data has been loaded.

To get to the deformable muscles, scroll to the bottom of the tree until you reach the "MuscleWrappers" node. It has sub-nodes *pelvis* and *right thigh*, which contain the individual muscles. After selecting them (clicking on them), the deformation is processed. However, you should choose the deformation method first. To do that select the root atlas node and click on the "vme" panel in area 2 (see figure A.1). There select the mass-spring method. You can also change the number of iterations that should be made and decides whether to use a constant or adaptive time step and change the value of the constant time step. An important checkbox is the "Use multiple objects" that is also present on the "vme" panel of the atlas. If it is selected, all muscles that are in the data set will be used during the

simulation. If it is unchecked, only the selected muscle will be used. To limit the simulation to only some muscles, check the “Use Multiple Objects” and “Enable filtering” checkboxes. Next, use the “Add muscle” or “Add region” buttons in area 3 (figure A.1) to add the individual muscles or whole regions (e.g. pelvis or right thigh) that you wish to exclude from the simulation.

After everything is set, click on the muscle wrapper of the muscle you want to deform. If you have the “Use multiple objects” selected, you may click any one muscle (that is not filtered) and all will be processed. You may choose different time from the time line at the bottom of the screen (area 4 in figure A.1). Note that this will have no effect if you are working with the rest pose atlas.

After the deformation has been processed (be aware that it might take up to several minutes), you can visualize the result. Click on the “View” menu and select “Add View -> Surface”. This will open a window inside the LHPBuilder which will visualize the data. Every object has a small box left from its name in the data tree. If you click on it, the object will be visualized in the surface window. To visualize the particles of the muscle, select the wanted muscle wrapper and then select “Create Particle” from the “Operations” menu, or use the shortcut CTRL+P. This will create a children node of the muscle wrapper that contains the particles and you may visualize it as any other node. The particles are visualized using small, fixed radiuses. The radiuses used for collision detection are different.

To rotate the view of the camera, click inside the window with the left mouse button and hold it while moving the mouse. If you want to move the camera in direction coinciding with the horizontal and vertical axes of the screen, hold the middle button instead. To move the camera in the direction perpendicular to the screen (“zoom”), hold the right button. Also, at the top of the view screen are buttons that will quickly move the camera to capture the selected object or the whole scene.

After selecting a muscle wrapper, numerous parameters can be changed in its “vme” panel. The panel is divided into several sections, each offering various control buttons. The most important ones for the mass-spring method are the following (sorted by the names of the sections in which they are found, starting from top):

- *Operational Mode:* use the checkbox “Generate fibers” to switch between the surface and the fibres of the muscle
- *Fibres Options:* the “method” drop-down menu chooses what method will be used for generating the fibres. To visualize the fibres that are based on the particles, select the last option – “update from particles”. The “Thick.” textbox select the thickness of the tubes used for visualization of the fibres. The “Num.” textbox contains the number of fibres and the “Res.” contains the number of particles per fibres. Use these two parameters to set the number of particles. However, note that the mass-spring method generates the particles on a basis of the fibres from the rest pose muscle. Therefore, if you want to use the “update from particles” method and you want to change the number of particles, you actually have to change it in the rest pose atlas, not the “atlas fused by walking”, and then start the deformation again. The recommended procedure for using the mass-spring method is: select the rest pose atlas and select the “Fast skinning” method (it is fastest and for the rest pose accurate). Then select

the muscle wrapper you wish to modify and set it up (select number of fibres, resolution, select either "simple slicing" or "advanced slicing" as the method for fibres). Then select the atlas fused by walking, choose the mass-spring method and set up the filters if you want. After that, click on the muscle wrapper you wish to deform using mass-spring method.

- *Particle Options:* the distance threshold decides how close a particle must be to a bone to be marked as fixed. The "type" drop-down box selects the spring layout and the "Num." textbox selects the number of neighbours to use if the nearest neighbour layout is selected. Note that just as with the previous options, these have to be changed for the rest pose muscle wrapper in order to be apparent in the muscle wrapper in the atlas fused by walking.
- *Decomposition:* if the "show constrained particles only" checkbox is selected, only the fixed particles will be visualized after the particle node have been created (CTRL+P).

Please keep in mind that by the time this thesis was released, the application was in the process of rapid development. Therefore, occasional crashes or some faulty functionality cannot be ruled out. Also, the storing of the refined data (described in section 4.4.2) currently does not work, so in fact, all the data are created in each simulation step. This unfortunately prolongs the processing significantly.

Programmer manual

The following text will briefly describe the structure of the source code related to the mass-spring method. The code is quite extensively commented in all needed parts. Therefore, this text will not explain every code snippet or class design in details. However, some concepts and techniques that might not be understandable on the first sight are explained.

The implementation was carried out as an extension to the medVMEMuscleWrapper class, which handles deformation of the muscles as well as fibre generation. This class is quite large, so it has been divided among several files based on the functionality. The core of the class, medVMEMuscleWrapper_Core.cpp, is responsible for the update of the muscle, therefore it contains the methods for deforming the muscle – *DeformMuscle()* – and for generating the fibres – *GenerateFibers()*. medVMEMuscleWrapper_Helper.cpp contains various support methods for storage and loading of the data. For the purposes of the mass-spring method, the most important part of the "Helper" is the method for particle creation – *GenerateParticles()* – and various associated methods that are called from it, like the methods generating the neighbourhood of particles on the basis of chosen layout, a method for creating fibres from the particles etc. Excluding the code for the spring layouts, the majority of the code associated with the mass-spring method that is in the "Helper" was written by Yubo Tao Ph.D.

To distinguish the deformation methods, the variable *m_DeformationMethod* is modified via the user interface. The deformation, i.e. the soft-body simulation, is processed in the *DeformMuscle()* method. The part of this method which is associated with the mass-spring approach (*m_DeformationMethod == 2*) proceeds as described in section 4.1. Most of the time it uses the code of the *vtkMassSpringMuscle* class, which can be found in the *vtkMassSpringMuscle.cpp/h* files. This file accumulates most of the code created by the author of this thesis, spread among four classes, which will be described in the following text: *SphereBoundingBox, vtkMSSDataSet, vtkMassSpringMuscle* and *vtkMassSpringBone.*

The *SphereBoundingBox* is the BVH used for collision detection. When creating the root bounding box, the *parent* and *spheres* parameters passed in the constructor should be set to NULL. After the constructor is called, use the method *SetContentFromArray* to set the input. This method is overloaded in order to enable usage of only selected primitives from among the whole content (the boundary particles) or all of it. The constructor with non-NULL parameters is used when creating the children boundary boxes. They do not need to set the content explicitly as it is given them from the parent node during splitting (*SplitRegularly()* method).

The *vtkMSSDataSet* is a simple data structure that encapsulates the "refined data" described in section 4.4.2. The reason why it is inherited from the *vtkPolyData* is to enable the storage of the structure via VME links (method *StoreMSSParticles* in the "Helper"). Using this method, the data can be saved and reused.

The *vtkMassSpringMuscle* class contains most of the functionality associated with the soft-body simulation. The *CreateData* method is responsible for creating the refined data. It takes the *m_Particles* vector as an input. This is a vector of all particles created in the "Helper" using the *GenerateParticles()* method. Each particle is described using the *CParticle* structure, which encapsulates the neighbours of the particle (i.e. the particles with which it should be connected by a spring), the position, if it is fixed and/or boundary particle etc. The *CreateData* method generally only transforms this information into structures that are more easily and more efficiently processed, which in most cases means one dimensional arrays of the individual qualities. The only information that is not in the *CParticle* structure is the radiuses and the closest particles to each vertex, which is used for the surface deformation. The *GenerateLinksUpdateRadiuses* does both of these (the setting of radiuses is bound to the distance to surface vertices, this way it does not have to be computed twice).

After the data are created, they should be stored using a VME link. However, at the time of releasing this thesis, there was an unresolved issue of memory leaks that stems from this storage system. Therefore, it is commented out in the code for now and the data are actually created in each simulation step anew.

The *MassSpringSystemCPU.cpp/h* contains the code of the actual MSS created by Zelený. Methods needed for the two-step processing (adaptive time steps, see section 5.3) were added to it. Therefore, if one wants to use a fixed time step, use the method *NextStep*. For the adaptive time step, use the methods *NextStepForces()* and *NextStepPositions()* in succession (in this order) and the *GetDt()/SetDt()* to set the time step in between the two calls. If the fixed time step is used, the variable *m_IterationStep* can be used to set how large the time step should be.

Overview of appended material

The appended DVD contains four directories with additional material:

- *Text:* this directory contains the text of this thesis in form of a PDF document for printing and the source MS Office Word document.
- *Source:* this directory contains the source code created for this thesis. However, not all parts of the LHPBuilder application are distributed as open source, therefore only several files with source code is appended, generally the code that is described in the Programmer manual above.

- *Demo:* an installation program for the demonstration version of the LHPBuilder is in this folder. The User manual describes how to install and use this program.
- *Test_results:* the source materials for graphs, tables and figures presented in chapter 5 can be found in this folder. Individual subfolders are named according the experiment they belong to, including the section number of the chapter in which they are described. Most of the results are in the form of MS Office Excel documents. The file “pattern.xls” describes the meaning of the data in each of the document using colour coding of individual cells. The names of the documents usually describe the parameter setting used for the given test.