FAST DEFORMATION FOR MODELLING OF MUSCULOSKELETAL SYSTEM

Josef Kohout[1], Petr Kellnhofer[1] and Saulo Martelli[2]
[1]*Department of Computer Science and Engineering, University of West Bohemia, Plzeň, Czech Republic*
[2]*Laboratorio di Tecnologia Medica, Istituto Ortopedico Rizzoli, Bologna, Italy*

Keywords: Deformation, Laplacian, Volume Preservation, Muscle Modelling.

Abstract: This paper proposes a gradient domain deformation for wrapping surface models of muscles around bones as they move during a simulation of physiological activities. Each muscle is associated with one or more poly-lines that represent the muscle skeleton to which the surface model of the muscle is bound so that transformation of the skeleton (caused by the movement of bones) produces transformation of the vertices of the mesh subject to Laplacian linear constraints to preserve the local shape of the mesh and non-linear volume constraints to preserve the volume of the mesh. All these constraints form a system of equations that is solved using the iterative Gauss-Newton method with Lagrange multipliers. Our C++ implementation can wrap a muscle of medium size in about a couple of ms up to 400 ms on commodity hardware depending on the type of parallelization, whilst it can keep the change in volume below 0.04%. A preliminary biomechanical assessment of the proposed technique suggests that it can produce realistic results and thanks to its rapid processing speed, it might be an attractive alternative to the methods that are used in clinical practise at present.

1 INTRODUCTION

The musculoskeletal modelling and simulation is an essential step in the process of looking for an optimal strategy to provide patients suffering from various musculoskeletal disorders, such as osteoporosis, with better healthcare. The most common and traditional models (e.g. (Garner and Pandy, 2000), (Delp and Loan, 2000)) assume that the muscle mechanical action occurs along a poly-line, namely the action line, joining origin and insertion points of the muscle, i.e., sites at which the muscle is attached to the bone by a tendon. Essentially, an action line is a representation of fibres in muscle-tendon unit.

When the muscle path is almost linear throughout a motion, the action line can be defined as a simple straight line, otherwise, the action line is considered to be a poly-line passing through a number of intermediate via points, fixed to the underlying bone, so as to describe the muscle path in a more realistic way (Jensen and Davy, 1975), (Arnold et al., 2000), (van der Helm and Veenbaas, 1991) or wrapping around wrapping surfaces, again fixed to the underlying bone to geometrically constrain the action line (Garner and Pandy, 2000), (Delp and Loan, 2000), (Marsden and Swailes, 2008), (AnyBody, 2010), (OpenSim, 2010) and (Audenaert and Audenaert, 2008). In the most sophisticated models of this kind, action lines are considered as elastic strings that can wrap automatically around multiple surfaces, known as obstacles, which might be a set of spheres, cylinders, ellipsoids or cones (Garner and Pandy, 2000), a set of parallel contours (Gao et al., 2002) or arbitrary volumetric data (Favre et al., 2010).

Although action line models are commonly used in clinical practise (thanks to their processing speed), in most of them an overestimation of predicted joint loads is observable (Erdemir et al., 2007) and it has been suggested that this overestimation is significantly influenced by typical underestimation of the muscle lever arm caused by the fact that these models assume that the muscle fibre length is uniform over the entire muscle bundle that is modelled, which is, however, an assumption not often fulfilled in reality.

There are also much more complex models that represent the muscles by 3D finite-element hexahedral mesh whose vertices move in reaction to the bones movement (Blemker and Delp, 2005). Each cell of the mesh contains information about the direction of the muscle fibres present in its volume by mapping a cubical template of fibre architecture into the muscle mesh. When the mesh changes, so do the

Kohout J., Kellnhofer P. and Martelli S..
FAST DEFORMATION FOR MODELLING OF MUSCULOSKELETAL SYSTEM.
DOI: 10.5220/0003811100160025
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2012), pages 16-25
ISBN: 978-989-8565-02-0

paths of the fibres, i.e., muscle volume is wrapped around bones as they move in this model. A good agreement was found comparing the model results with static MRI images taken in different postures and it was showed that the muscle fibre deformations are highly non-uniform over the muscle volume. Finite-element based techniques, however, require large computational time and, therefore, they are not suitable for clinical practise.

In our previous work (Kohout et al., 2011), we attempted to bridge the gap between both approaches proposing a musculoskeletal model that is fast enough for use in clinical practise and for which it is reasonable to expect that the prediction of the muscle lever arm and the actual fibre length will be somewhere in between predictions by action-line methods and more accurate finite-element methods. In this model, a muscle is represented by a chaff of muscle fibres that are automatically generated in the volume defined by its surface mesh which itself automatically wraps around bones as they move. The wrapping process is based on the mesh skinning technique described by (Blanco and Oliveira, 2008) that is combined with a post-adjustment of mesh vertices to minimise the changes in volume following the idea presented by (Aubel and Thalmann, 2000). The decomposition of wrapped volume into muscle fibres is done by a slice-by-slice morphing of predefined fibres template into the wrapped muscle exploiting mean value coordinates (Ju et al., 2005). A muscle of medium size can be processed in a fast speed of one second on commodity hardware, however, for some muscles, the maximal loss in volume exceeded even 13%, which is above the physiologically acceptable limit (6% according to (Aubel and Thalmann, 2000)),

In this paper, we propose a new fast deformation method for wrapping of muscles that overcomes this problem as it can guarantee that changes in volume do not exceed the given threshold. This method is based on that described by (Huang et al., 2006) where volume preserving deformation is achieved by forming a system of non-linear equations that is solved by an iterative Gauss-Newton method with Lagrange multipliers. The main changes done to the original description were to adapt conditions and the computation process to our input data and goals. We also introduced progressive steps in iterations to speed up the convergence and parallelized the computation using OpenMP and, furthermore, investigated the possibility to parallelize the computation using GPU.

The remainder of this paper is structured as follows. In the next section, we give a brief survey of existing deformation methods that deal with volume preservation. Section 3 brings an overview of our method; details are described in sections 4 and 5. Section 6 presents the experiments that were performed. Section 7 concludes the paper and provides an overview of possible future work.

2 RELATED WORK

Deformation methods can be divided into three main categories: finite element methods (FEM), mesh-skinning techniques and gradient domain deformation techniques. FEM methods decompose the object to be deformed into a set of interconnected nodes that are scattered in the volume of the object (typically, they form tetrahedral or hexahedral meshes) and associate each node with a strain tensor for modelling local deformations at the node (Debunne et al., 2001). Although these methods produce accurate results, especially, when large numbers of nodes are used, they consume lot of memory and require substantial computations, which renders them unsuitable for large-scale real-time simulations.

Volume preserving mesh-skinning techniques usually compute automatically a displacement field that is used in the post-processing to shift vertices of the surface mesh being deformed outward the skeleton of the object. Most of these method focus on preserving volume locally (Botsch and Kobbelt, 2003), (Rohmer et al., 2008). A global volume correction technique is proposed by von Funck et al. (Von Funck et al., 2008). The method, however, cannot preserve shape of the mesh and avoid self-intersections. It is important to highlight that calculation of correct displacements requires that the skeleton lies inside the volume defined by the surface, which is a condition that is often violated in context of our application.

A good survey of gradient domain deformation techniques is given in (Xu and Zhou, 2009). For a surface mesh, these techniques define its energy as a function that contains terms for detail and volume preservation, position and other constraints. External forces at vertices are also expressed as an energy function. A product of these two functions gives the deformation energy function whose minimisation, which is done by solving an over-constrained system of linear or non-linear equations, yield the new positions of vertices in the deformed mesh (Huang et al., 2006), (Zhou et al., 2005). We note that gradient domain deformation techniques are usually designed for direct mesh editing, where only a small part of surface is subject to external forces, which is again something uncommon in our context.

3 METHOD OVERVIEW

Our method is based on the gradient domain deformation technique described in (Huang et al., 2006). In contrast to original description, however, it does not assume that the mesh skeleton, whose change triggers the deformation of surface mesh, is fully inside the volume of the mesh. In our case, the skeleton is a set of action lines representing the muscle (it is inherited from an action-line musculoskeletal model), i.e., it is a set of independent poly-lines describing the general path of muscle and being fixed to the underlying moving bone (represented also as surface mesh). As it can be seen in Figure 1, they typically do not fully lie inside the muscle volume. Many of them are just straight lines, so it was necessary to introduce some reference point (we use the centroid of the bone along which the muscle runs) to avoid unnatural rotation of the deformed muscle. It is also typical that poly-lines of the skeleton in its rest-pose (obtained from MRI data) and in its current-pose (defined by motion capture data) have nothing in common since the coordinate space of the motion data used to fuse the static rest-pose musculoskeletal model typically differs from the coordinate space in which this model was created. This is again something not considered in the original technique since it was designed for interactive animation. For more details, see our paper (Kohout et al., 2011).

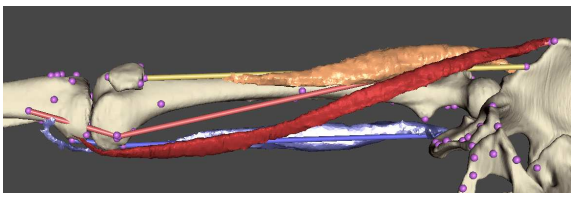

Figure 1: Sartorius, Rectus Femoris and Semitendinosus muscles and their action lines (in similar colours).

With the triangular surface mesh of the muscle in its rest-pose on input, the method starts with the construction of a very low resolution of this surface, which has only a few hundreds vertices but preserves the basic shape of muscle. It can be done using any decimation technique, however, it is preferable if the new mesh is outer envelope of the original one. Since muscles should be of simple shape, we use edge collapse technique (Hoppe, 1999) available in VTK toolkit (Schroeder et al., 2004) followed by an enlargement of the produced mesh by a slight shifting of the vertices in direction of their normals. Once the coarse mesh is ready, the method uses mean value coordinates interpolation (Ju et al., 2005) to express the coordinates of the original mesh vertices as a combination of the control coarse mesh vertex coordinates. We note that both the coarse mesh construction and the computation of mean value coordinates can be done (and it is done in our implementation) in the preprocessing to speed up the deformation process.

A linear condition to preserve the shape of the coarse mesh is formulated (details are given in Section 4). A relative position of the coarse mesh to the muscle skeleton in its rest-pose position is evaluated (using mean value coordinates interpolation) and a linear condition to preserve this position is formulated. Naturally, both conditions hold for the Cartesian coordinates $[x,s]$, where x are coordinates of vertices of the coarse mesh and s coordinates of the rest-pose poly-lines of the muscle skeleton. The deformation energy function, which is defined by these conditions, is zero for these coordinates, i.e., $f(x,s) = 0$. Conditions are, however, violated for $[x,s']$, where s' are coordinates of the vertices of current-pose poly-lines, i.e., $f(x,s') > 0$. The aim is to find new coordinates x' of the coarse mesh vertices so that $f(x',s') = min$, whilst the volume of the coarse mesh is preserved, i.e., $g(x) = c = g(x')$, where $g(x)$ is a non-linear function calculating the volume of the object from its surface coordinates x.

The solution to this problem lies in solving an over-constrained system of linear equations (it is defined by the linear conditions) that is subject to the non-linear volume constraint. The iterative Gauss-Newton method with one Lagrange multiplier λ is used for this purpose. As the coarse mesh contains a few hundreds of vertices only, the numerical stability of this method is good and, furthermore, it converges quickly. When the speed of convergence drops below the predefined threshold (or after some large number of iterative steps), we switch the deformation process to the next stage. In this stage, in every step of the Gauss-Newton method, the next value of Lagrange multiplier λ is not calculated to preserve the volume of the coarse mesh but to preserve the volume of muscle mesh, which means that the deformed muscle mesh must be reconstructed from the current coarse mesh (coordinates x') using the precomputed mean value coordinates and the difference in volume of the deformed and original muscle mesh evaluated. The process stops when the difference is below the given threshold (or alternatively after some large number of iterations).

The overview of the method is in Figure 2.

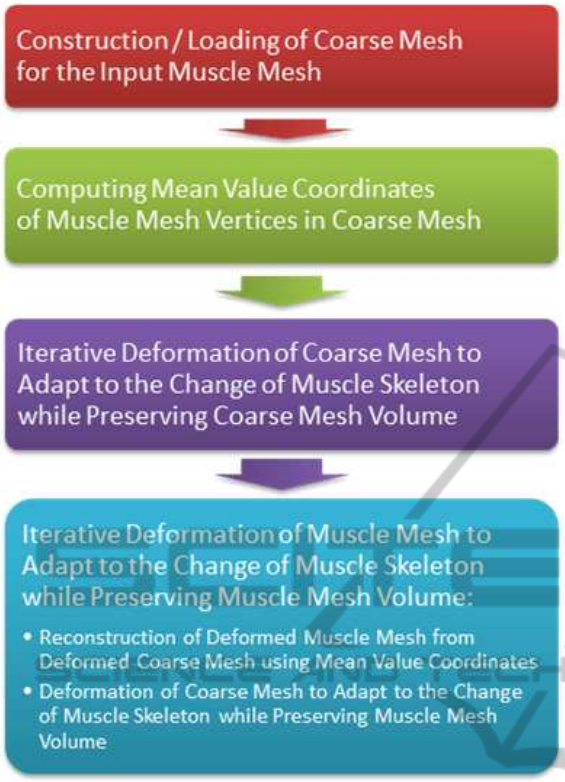

Figure 2: The schema of our deformation.

4 CONDITIONS

We can divide conditions into soft and hard conditions. While the soft conditions, which are the condition of Laplacian to preserve the shape of the model to be deformed and the condition of skeleton to initiate the deformation process, might not be fulfilled, i.e., their error (deformation) energy might not be non-zero after the deformation, the hard conditions must be met. In our case, we would like to preserve the original volume, so its value does not change.

4.1 Condition of Laplacian

Laplacian (Nealen et al., 2006) is a very common term in computer graphic. It is used to describe a local shape of mesh as a relation between each vertex x_i and its one-ring neighbours Ω, which is expressed as the difference between the weighted mean of these neighbours and the vertex, i.e., $\delta(x_i) = (\sum_{j\in\Omega} w_{i,j} \cdot x_j) - x_i$. Using cotangent weights ω_j, which are calculated as $\cot|\angle x_i x_{j+1} x_j| + \cot|\angle x_i x_{j-1} x_j|$, ensures that the shape of the mesh will be kept similar to the original one as much as possible.

The formula for the Laplacian can be rewritten as $\delta(x_i) = \sum_{j=0}^{N-1} w_{i,j} \cdot x_j$, where N is the number of vertices. Weights $w_{i,j\neq i}$ for not connected vertices x_i and x_j are zero. The weight $w_{i,j=i} = -1$. With this, the set of equations for the Laplace condition can be easily formed as $\mathcal{L} \cdot x = \delta(x)$, where $\mathcal{L}$ is a sparse matrix of weights $w_{i,j}$ for one vertex on each row, x is the $N \times 3$ matrix of Cartesian coordinates of vertices and the matrix $\delta(x)$ on the opposite side of equation contains Laplacian differential vertices.

The differential vertices are, however, not invariant to the rotation of the mesh. Ignoring this fact, may lead into flattening or unnatural rotation of the model. We prevent this by rotating the differentials $\delta(x)$ around axis ρ by angle ϕ computed in such a manner that vector $n = \frac{u\times v}{|u\times v|}$ after being rotated around this axis ρ by angle ϕ is identical with vector $n' = \frac{u'\times v'}{|u'\times v'|}$. Vectors u, v are calculated as $u = s_1 - s_0, u' = s'_1 - s'_0$ and $v = r - s_0, v' = r' - s'_0$, where s_i are vertices of the muscle skeleton in its rest-pose position and s'_i are the corresponding vertices of this skeleton in the current pose position, r is the rest-pose reference point chosen as the centroid of the bone along which the muscle should wrap, whilst r' is the corresponding current-pose reference point.

4.2 Condition of Skeleton

The condition of skeleton is used to preserve the relative position of the mesh and its skeleton. First, it is necessary to match the poly-lines representing the rest-pose and the current-pose paths of the skeleton since these poly-lines may not be of the same length or even may not have the same number of points. The matching process (for detail, see (Kohout et al., 2011)) exploits an arc-length parameterization of both poly-lines to successively introduce vertices from the current-pose poly-line into the rest-pose one (and vice versa), which is followed by subdividing long segments. By the end of this process, both poly-lines will have the same number of points.

For each vertex s_i of the rest-pose poly-line, its mean value coordinates (MVC) $k_{i,j}$ in the mesh are computed (Ju et al., 2005), so that $s_i = \sum_{j=0}^{N-1} k_{i,j} \cdot x_j$. With this, the set of equations for the condition of skeleton can be easily formed as $\mathcal{S} \cdot x = s'$, where $\mathcal{S}$ is a matrix of MVC coordinates $k_{i,j}$ for one skeleton vertex on each row, x is the $N \times 3$ matrix of Cartesian coordinates of mesh vertices and the matrix s' on the opposite side of equation contains Cartesian coordinates of the current-pose poly-line vertices.

4.3 Condition of Volume

Since we demand preserving the original volume as precisely as possible, the condition of volume must be formulated. It is simply defined as the non-linear equation $v(x) - c = 0$, where $v(x)$ is the function evaluating the volume of the mesh from its vertices x, whilst c is the original volume of the mesh. The volume of the closed mesh can be evaluated in linear time as a sum of oriented volumes of tetrahedrons defined by each mesh triangle and some reference point, e.g. $(0,0,0)$. Hence, $v(x) = \frac{1}{6}|\sum_{i=0}^{N_t-1}(x_{i_1} \times x_{i_2}) \cdot x_{i_3}|$, where N_t is the number of triangles in the mesh and $x_{i_1}, x_{i_2}, x_{i_3}$ are coordinates of vertices of i-th triangle. We note that the condition of volume is a hard constraint, i.e., it must be always fulfilled. This complicates the solving process, indeed.

5 SOLVING

5.1 Iterative Algorithm Construction

We have now a set of linear conditions in form of

$$\begin{pmatrix}\mathcal{L}\\ \mathcal{S}\end{pmatrix} \cdot x = \begin{pmatrix}\delta(x)\\ s'\end{pmatrix} \equiv A \cdot x = d(x) \qquad (1)$$

where $d(x)$ is a function of x because the differential vertices $\delta(x)$ are not rotation invariant and, therefore, depends on the Cartesian coordinates of x. However, thanks to rotation of the differential vertices prior to solution (see Section 4.1), the dependence of these vertices on rotation is negligible, so we can approximate this function by constant matrix d.

The obtained linear system is overdetermined, so we have to use least squares method and solve

$$A^T \cdot A \cdot x = A^T \cdot d \equiv L \cdot x = b \qquad (2)$$

with the hard constraint for volume preservation:

$$g(x) = v(x) - c = 0 \qquad (3)$$

We will use an iterative approach as follows. We define function $f(x)$ as

$$f(x) \equiv L \cdot x - b \qquad (4)$$

that we will minimise using *Gauss-Newton* numerical method. Starting with an initial solution x_0 (the original Cartesian coordinates of the mesh to deform in our case), this method successively improves this solution by adding an increment h_k to it in every step, i.e., $x_{k+1} = x_k + h_k$, where h_k successively decreases down to zero, i.e., for $k \to \infty$, $h_k \to 0$.

The change of f after every step h_k added to x_k can be described linearly using the first derivative as

$$f(x_{k+1}) \equiv f(x_k + h_k) \approx f(x_k) + J_f(x_k) \cdot h_k \qquad (5)$$

where $J_f(x_k)$, or in shorter form J_f, is Jacobi matrix of $f(x_k)$. Now, we can derive the formula for h_k:

$$\begin{gathered} f(x_k + h_k) = 0 \\ f(x_k) + J_f \cdot h_k = 0 \\ f(x_k) = -J_f \cdot h_k \\ {J_f}^T \cdot f(x_k) = -{J_f}^T \cdot J_f \cdot h_k \\ ({J_f}^T \cdot J_f)^{-1} \cdot {J_f}^T \cdot f(x_k) = h_k \\ h_k = ({J_f}^T \cdot J_f)^{-1} \cdot {J_f}^T \cdot f(x_k) \qquad (6) \end{gathered}$$

Following the description in (Li, 1994), we add volume loss compensation to every step, which yields the new equation:

$$h_k = -({J_f}^T \cdot J_f)^{-1} \cdot ({J_f}^T \cdot f(x_k) + {J_g}^T \cdot \lambda_k) \qquad (7)$$

where J_g is Jacobi matrix of $g(x_k)$ and λ_k is the Lagrange coefficient. The matrix J_g, which is composed from partial derivatives of the volume function $v(x_k)$, describes the directions of volume growth, whilst the coefficient λ_k describes its amount.

The Lagrange coefficient λ_k must be chosen to both fix the overall volume loss which has already occurred (in previous steps) and to compensate the volume change which will be caused by adding h_k from eq. 6 to the mesh in the current step. This means that λ_k depends on the volume function $g(x_k)$ and the volume change $h_k \cdot J_g$ (when $h_k \to 0$), i.e.,

$$\lambda_k = R_k \cdot (g(x_k) + J_g \cdot ({J_f}^T \cdot J_f)^{-1} \cdot J_f^T \cdot f(x_k)) \qquad (8)$$

where R_k, which compensates $({J_f}^T \cdot J_f)^{-1}$ from eq. 7 and normalises J_g, is calculated as

$$R_k = (J_g \cdot ({J_f}^T \cdot J_f)^{-1} \cdot {J_g}^T)^{-1} \qquad (9)$$

If $|h_k|$ is smaller than the given ε (or after some sufficient large amount of steps), the deformation process stops yielding the Cartesian coordinates x' of the deformed mesh in x_k.

5.2 Improvements of the Algorithm

Huang et al. (Huang et al., 2006) suggested changing the length of step h_k by multiplying it by some parameter α, whose values ranges from 0 to 1, to improve the stability of the process. Hence, the deformation formula changes to

$$\begin{gathered} x_{k+1} = x_k + \alpha \cdot h_k \\ x' = x_n \Leftrightarrow h \to \varepsilon \end{gathered} \qquad (10)$$

We have found experimentally that instead of using a constant value for this parameter, it is better to iteratively decrease the α value (starting from value 0.5)

Table 1: Dependence of the number of iterations and the volume preservation ratio on the α values.

α	Num. of Iterations	Vol. preservation
0.025	186	99.89%
0.050	106	99.85%
0.100	59	99.77%
0.250	27	99.58%
0.500	24	99.35%
0.750	76	99.19%
adaptive	57	99.93%

while getting closer to the final solution. This leads to a faster convergence at the beginning and a more precise solution at the end – see Table 1.

However, when we shorten the step, we shorten also the compensation of the cumulative volume error. This is an undesired effect, so we have to divide the amount of volume compensation in the step h_k by α_k, which will disable the effect of α_k multiplication in later steps and allow all the volume error to be fixed.

Another issue is that processing of large meshes would be not only time and memory consuming but also numerically unstable. Hence, as described in section 3, instead of using the original mesh, the iterative method works with its coarse version; the coordinates of the original mesh can be obtained from the current coarse mesh x_k by MVC coordinates interpolation. As a consequence of this, to preserve the volume of the original mesh in the deformation, the volume function v (see eq. 3) must be evaluated for the original high detail mesh. In practise, however, we do not need to evaluate the volume so precisely at the beginning, which means that we can use the coarse mesh for the volume evaluation as well and only after getting close to the solution, we switch to the original mesh. As the coarse mesh has larger volume than the original one (this is caused by the fact that the coarse mesh is an external envelope of the original mesh), which means that absolute size of volume error will be higher, when using the coarse mesh for the calculation of volume, the evaluated volume must be divided by the ratio of original volumes of those two meshes. The final set of solution equations then is:

$$\begin{aligned}
&x_{k+1} = x_k + \alpha_k \cdot h_k \\
&h_k = -({J_f}^T \cdot J_f)^{-1} \cdot ({J_f}^T \cdot f(x_k) + {J_g}^T \cdot \lambda_k) \\
&\lambda_k = R_k \cdot (dv(x_k)/\alpha_k + J_g \cdot ({J_f}^T \cdot J_f)^{-1} \cdot J_f^T \cdot f(x_k)) \\
&R_k = (J_g \cdot ({J_f}^T \cdot J_f)^{-1} \cdot {J_g}^T)^{-1} \\
&dv(x_k) = \{ \begin{array}{l} g(x_k) \cdot \frac{g(x_0)}{g(\tilde{x_0})}; \text{ if } h_k > \varepsilon \\ g(\tilde{x_k}); \text{ if } h_k \to \varepsilon \end{array}
\end{aligned} \tag{11}$$

where x_k is the matrix of the coarse mesh vertices and $\tilde{x_k}$ is the matrix of the original mesh vertices.

5.3 Evaluation of Equations

Although all the relations have been formed, it may still not be clear how to use them.

Matrix A from eq. 1 is composed from the Laplacian and skeleton matrices (see section 4. These matrices were built considering x to be $N \times 3$ matrix with all three single vertex coordinates on one row, where N denotes the number of mesh vertices. With this, the matrix $L = A^T \cdot A$ is $N \times N$, which is an important memory reduction in comparison to obvious $3 \cdot N \times 3 \cdot N$ (we have N conditions for x-coordinate, N for y-coordinate and N for z-coordinate).

Matrix $d \approx d(X)$ has the same width as x and the same height as A, i.e., $b = A^T \cdot d$ is a matrix $N \times 3$. Jacobi matrix $J_f(x)$ of $f(x) = L \cdot x - b$ is the first derivative of linear function. Hence,

$$J_f(x) = L \tag{12}$$

and the size of $J_f(x)$ is $N \times N$.

The volume error function $g(x)$ from eq. 3 is a scalar function of $N \times 3$ mesh coordinates. Jacobi matrix of $J_g(x)$ is a row vector $1 \times 3 \cdot N$ of partial derivatives of $g(x)$ with respect to each coordinate, so $J_g(x) = (\frac{\delta g(x)}{\delta x_{0_x}}; \frac{\delta g(x)}{\delta x_{0_y}}; \frac{\delta g(x)}{\delta x_{0_z}}; \frac{\delta g(X)}{\delta x_{1_x}}; \frac{\delta g(x)}{\delta x_{1_y}}; \frac{\delta g(x)}{\delta x_{1_z}}; \cdots)$. This can be done analytically:

$$\frac{\delta g(x)}{\delta x_i} = \frac{1}{6} \sum_{j=1}^{N_i} (x_{j_1} \times x_{j_2}) \tag{13}$$

where N_i is the number of triangles sharing the vertex x_i and x_{j_1} or x_{j_1} are the other vertices of j-th triangle.

Because of compression used for L and x, the sizes of matrices that must be multiplied to get λ_k and h_k (see eq. 11) are incompatible. Hence, it is necessary either to expand matrices L and J_f to $3 \cdot N \times 3 \cdot N$ form or to use a little tricky manipulations with rows and columns of operands as follows. First, we transform ${J_g}^T$ from a vector to $N \times 3$ matrix such that partial derivatives of the same mesh vertex are on a single row of the matrix. This allows us to multiply $({J_f}^T J_f)^{-1} {J_g}^T$. Now, we must transform the result matrix back into a vector. The dot product of J_g and this vector gives as a scalar value whose reciprocal is R_k. A similar solution is used for the right part of formula for λ_k (eq. 11) and in evaluation of formula for h_k.

The simplified version of the algorithm in C is:

```
change = DOUBLE_MAX;
alpha = 0.1;
X[0] = orig;
i = 0;
while (change > epsilon) {
  g = calculateVolumeError(orig, X[i]);
  Jg = calculateJg(X[i]); // eq. 13
  lambda = .. //eq. 11, Jf = L
```

```
  h = ... //eq. 11
  X[i+1] = X[i] + alpha * h;
  change = |X[i+1]-X[i]|
  if (change < thrs) {
    //we are getting close to the solution
    change = decreaseAlpha();
  }
  i++;
}

return X[i+1];
```

6 EXPERIMENTS AND RESULTS

Our approach was implemented in C++ (MS Visual Studio 2010) under the Multimod Application Framework – MAF (Viceconti et al., 2004), which is a visualisation system based mainly on VTK (Schroeder et al., 2004) and other specialised libraries. This framework is designed to support the rapid development of biomedical software. It is particularly useful in multimodal visualisation applications, which support the fusion of data from multiple sources and in which different views of the same data are synchronised, so that when the position of an object changes in one view, it is updated in all the other views. Our implementation was then integrated into the MuscleWrapping software [1] that is being developed within the VPHOP project (VPHOP, 2010). Experiments were conducted on various real data sets of muscles with typical sizes about 15K vertices on Intel Core i7 2.67 GHz, 12 GB DDR3 1.3GHz RAM and Intel Core i7 4.4 Ghz, 16 GB DDR3 1.6GHz RAM, both with Windows 7 Pro x64.

Figure 3 shows the deformation of muscles of right thigh at current-pose frames t = 0.00, 0.25, 0.50 and 0.75 of the 1.56 seconds long walk sequence. The deformation process of a single muscle consumed 250–650 ms per frame on Intel Core i7 2.67 GHz computer, which makes the new method to be about 8–20 times slower than our former deformation method that was based on the mesh skinning with volume correction in post-processing (Kohout et al., 2011). Nonetheless, the proposed method is still fast enough for our purpose since the deformation of all these muscles required a few seconds. We note that this time does not include the time consumed in the pre-processing to compute the coarse mesh and to establish the relationship between the both meshes (our implementation takes up to one minute depending on the model complexity).

[1] http://graphics.zcu.cz/Projects/Muskuloskeletal-Modeling

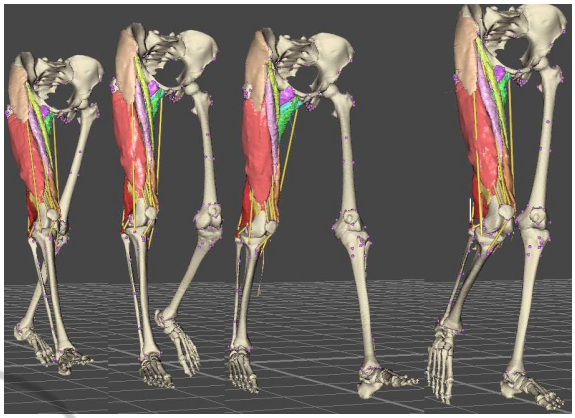

Figure 3: Wrapping of the muscles of right thigh during the movement.

The shapes of the deformed and rigidly transformed sartorius muscle are compared in Figure 4. While the rigid transformation does not preserve the attachment of the muscle to the bones and provides the user with an unnatural result, the deformation produces a plausible result. The loss in the volume of the muscle was about 0.02%, which is much lower than the loss 11% observable in our former method.

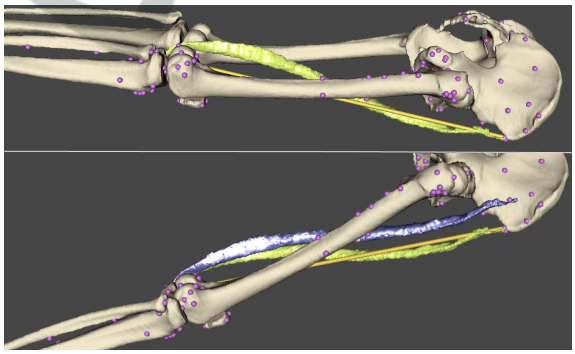

Figure 4: Sartorius muscle with its action line (yellow) in the rest-pose position (above) and at frame t = 0.00 of the walk sequence (below), transformed rigidly (blue) and deformed (green) according to the change of its action line.

As it can be seen in Figure 5, the new method preserves the volume much better also for other muscles and other poses. Nevertheless, for some poses, the maximal change in the volume of muscle may exceed 6%, which is, according to (Aubel and Thalmann, 2000)), the physiologically acceptable limit. The reason for such a poor result is that the iterative process terminated for semitendinosus muscle not because the optimal solution had been reached but because the maximal allowed number of iterations (100 in our case) had been performed. For all other mus-

cles, the maximal volume error is within the physiologically acceptable limits.

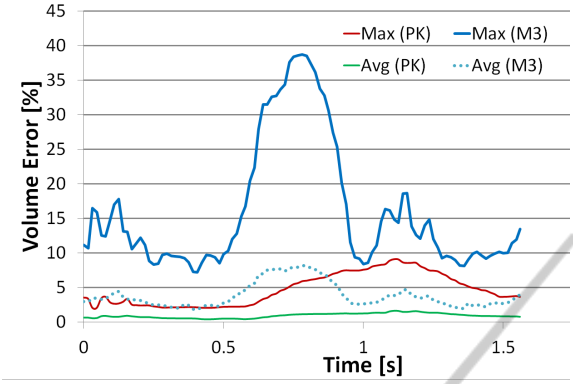

Figure 5: Comparison of maximal and average volume errors for muscles of the right thigh in the walk sequence for our former method (M3) and the proposed one (PK).

The results can be easily improved by increasing the maximal allowed number of iterations as it is demonstrated in Figure 6 that shows the dependence of the deformation energy and volume error on the number of iterations for semitendinosus muscle. More than 170 iterations were needed before the deformation energy dropped to the level when the method could start using the detailed mesh for the volume computation – see the abrupt change in the volume error function. Whilst the volume error was almost 10% after 100 iterations (see Figure 5), the final volume error (after 200 iterations) is about 0.04%.

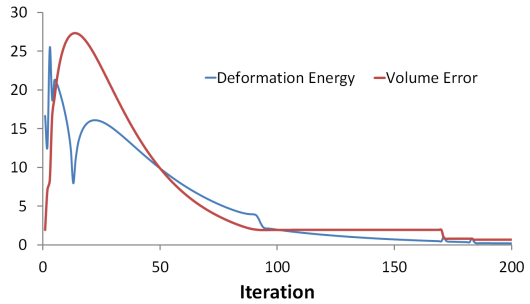

Figure 6: Monotonic functions of the deformation energy and the volume error in the dependence on the number of iterations.

There are two main factors contributing to such a slow convergence of the method. First, it has been found that many tested muscles (semitendinosus muscle included), extracted from MRI images during our previous EU-funded project (LHDL, IST-2004-026932) as described in (Testi et al., 2010), are not closed manifolds. This causes troubles when creating the coarse mesh and expressing the relations between both meshes, which lead into an appearance of ill-shaped triangles in the coarse mesh, and, if non-manifold edges are present at a very narrow place (typically, tendon part), also into having more than just one component of the coarse mesh. As a result, the method needs more iterations and, furthermore, strange spikes on the deformed mesh may appear. This could be solved by refining the muscles, however, this is a part of our future work.

Next, for some muscles (e.g., sartorius), there is only a loose relation between the muscle surface and its skeleton, which negatively influences the number of iterations required. This is demonstrated in Table 2 that shows the processing time needed for various – already refined – muscles of different sizes. Vastus lateralis, which contains more than 50K triangles but have skeleton going through its volume, was deformed faster than sartorius, which has less than 10K triangles but its skeleton lies outside the mesh.

Table 2: Performance of our method on Intel Core i7 2.67 GHz, 12 GB DDR3 1.3GHz RAM.

Muscle	Num. of Triangles	Time [ms]
Rectus femoris	5238	268
Sartorius	9908	476
Adductor longus	13912	550
Vastus lateralis	50240	379

To reduce the required processing time, we parallelized routines for the multiplication of matrices, the calculation of MVC coordinates of the muscle mesh in the coarse mesh and the reconstruction of the muscle mesh from the coarse one using OpenMP, a multiplatform interface for parallel programming in C, C++ and Fortran described, e.g., in (Gatlin and Isensee, 2010). OpenMP allows automatic assignment of independent iterations of a for cycle to working threads, providing that the cycle is prefixed in the code by special directives as it is shown in Figure 7.

Table 3 shows the results of the parallelization. The optimal performance obviously is achieved when the number of threads to be used for the calculation equals the number of physical cores of CPU. As it can be seen, speed-up of at least 40% can be reached on commodity hardware.

We have investigated also an option of speeding up multiplication of matrices by employing GPU using CUBLAS library, which is a standard part of CUDA system (nVidia, 2011). Providing that matrices to be multiplied are represented by one dimensional arrays, the speed-up on a system with NVIDIA GeForce 9600 GT 512 MB is about 978, otherwise,

```
#pragma omp parallel private(i, j) shared(A, Bt, widthA, width, height)
{
  #pragma omp for
  for (i = 0; i < height; i++)
  {
    double *rowA = A->values[i];
    double *resultCell = result->values[i];
    double **pointerBt;

    for (j = 0, pointerBt = Bt->values; j < width;
      j++, pointerBt++, resultCell++)
    {
      (*resultCell) = PKUtils::DotN(rowA, *pointerBt, widthA);
    }
  }
}
```

Figure 7: Parallelization of the multiplication of matrices A and B by OpenMP directives).

Table 3: Time needed by parallel version for processing a mesh of 6978 vertices on Intel Core i7 4.4 Ghz.

Threads	Time [ms]	Speed-up
1	397	–
2	278	1.43
4	248	1.60
8	262	1.51
16	295	1.35

which is our case, it is only 2.26 because of the required transformation from the internal data structure into the array and vice versa. Still this is an important speed-up.

7 CONCLUSIONS

In this paper we described a gradient domain volume preserving deformation method suitable for muscle wrapping application. The deformation is governed by the change of the action line (skeleton) of the muscle mesh - the method iteratively alters the positions of mesh vertices to adapt the change of the skeleton whilst preserving the volume. For any of tested muscles, 200 iterations are enough to achieve volume error below 0.04%, 100 iterations are sufficient for any manifold muscles. The sequential processing of typical mesh requires about 400 ms on commodity hardware. When parallelized using OpenMP the required time may drop to 250 ms depending on hardware configuration and even to a couple of ms (estimated) when parallelized using CUDA. Hence, the method is fast enough to be used for the muscle wrapping for clinical practise. An understandable drawback of the approach is its sensitivity to the mesh quality. For non-manifold meshes, which often result from marching cube iso-surface extraction, the convergence of the method may be very slow or even the method may produce a mesh having some parts unnaturally deformed (e.g., spikes appearance, strange twists). Some mesh filtering prior to the deformation is, therefore, typically needed. This filtering is a part of our future work as well as proper validation of our method in clinical context.

ACKNOWLEDGEMENTS

This work was supported by the Information Society Technologies Programme of the European Commission under the project VPHOP (FP7-ICT-223865). The authors would like to thank the various people who contributed to the realisation of the MAF and LHPBuilder software and to various people who provided condition under which the work could be done.

REFERENCES

AnyBody (2010). Anybody technology, http://www.anybodytech.com.

Arnold, A. S., Salinas, S., Asakawa, D. J., and Delp, S. L. (2000). Accuracy of muscle moment arms estimated from mri-based musculoskeletal models of the lower extremity. *Computer aided surgery official journal of the International Society for Computer Aided Surgery*, 5(2):108–119.

Aubel, A. and Thalmann, D. (2000). Efficient muscle shape deformation. In *IFIP*, pages 132–142.

Audenaert, A. and Audenaert, E. (2008). Global optimization method for combined spherical-cylindrical wrapping in musculoskeletal upper limb modelling. *Computer Methods and Programs in Biomedicine*, 92(1):8–19.

Blanco, F. R. and Oliveira, M. M. (2008). Instant mesh deformation. *Proceedings of the 2008 symposium on Interactive 3D graphics and games SI3D 08*, 1(212):71–78.

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

Botsch, M. and Kobbelt, L. (2003). Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum*, 22(3):483–491.

Debunne, G., Desbrun, M., Cani, M.-P., and Barr, A. H. (2001). Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 31–36, New York, NY, USA. ACM.

Delp, S. L. and Loan, J. P. (2000). A computational framework for simulating and analyzing human and animal movement. *Computing in Science and Engineering*, 2(5):46–55.

Erdemir, A., McLean, S., Herzog, W., and Van Den Bogert, A. J. (2007). Model-based estimation of muscle forces

exerted during movements. *Clinical Biomechanics*, 22(2):131–154.

Favre, P., Gerber, C., and Snedeker, J. G. (2010). Automated muscle wrapping using finite element contact detection. *Journal of Biomechanics*, 43(10):1931–1940.

Gao, F., Damsgaard, M., Rasmussen, J., and Christensen, S. T. (2002). Computational method for muscle-path representation in musculoskeletal models. *Biological Cybernetics*, 87(3):199–210.

Garner, B. and Pandy, M. (2000). The obstacle-set method for representing muscle paths in musculoskeletal models. *Computer Methods in Biomechanics and Biomedical Engineering*, 3(1):1–30.

Gatlin, K. S. and Isensee, P. (2010). Reap the benefits of multithreading without all the work. http://msdn.microsoft.com/en-us/magazine/cc163717.aspx.

Hoppe, H. (1999). New quadric metric for simplifying meshes with appearance attributes. *Proc IEEE Conference on Visualization VIS99*, pages 59–66.

Huang, J., Shi, X., Liu, X., Zhou, K., Wei, L.-Y., Teng, S.-H., Bao, H., Guo, B., and Shum, H.-Y. (2006). Subspace gradient domain mesh deformation. *ACM Transactions on Graphics*, 25(3):1126–1134.

Jensen, R. and Davy, D. (1975). An investigation of muscle lines of action about the hip: A centroid line approach vs the straight line approach. *Journal of Biomechanics*, 8(2):103–110.

Ju, T., Schaefer, S., and Warren, J. (2005). Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics*, 24(3):561–566.

Kohout, J., Clapworthy, G. J., Martelli, S., Wei, H., Viceconti, M., and Agrawal, A. (2011). Fast muscle wrapping. *Computers & Graphics*. Submitted for publication.

Li, S. Z. (1994). Markov random field models in computer vision. In Eklundh, J.-O., editor, *ECCV (2)*, volume 801 of *Lecture Notes in Computer Science*, pages 361–370. Springer.

Marsden, S. P. and Swailes, D. C. (2008). A novel approach to the prediction of musculotendon paths. *Proceedings of the Institution of Mechanical Engineers Part H Journal of engineering in medicine*, 222(1):51–61.

Nealen, A., Igarashi, T., Sorkine, O., and Alexa, M. (2006). Laplacian mesh optimization. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, GRAPHITE '06, pages 381–389, New York, NY, USA. ACM.

nVidia (2011). Cuda zone. http://developer.nvidia.com/category/zone/cuda-zone.

OpenSim (2010). Opensim project, https://simtk.org/home/opensim.

Rohmer, D., Hahmann, S., and Cani, M.-P. (2008). Local volume preservation for skinned characters. *Computer*, 27(7):1919–1927.

Schroeder, W., Martin, K., and Lorensen, B. (2004). *The Visualization Toolkit, Third Edition*. Kitware Inc.

Testi, D., Quadrani, P., and Viceconti, M. (2010). Physiomespace: digital library service for biomedical data. *Philos Transact A Math Phys Eng Sci*, 368(1921):2853–2861.

van der Helm, F. and Veenbaas, R. (1991). Modelling the mechanical effect of muscles with large attachment sites: Application to the shoulder mechanism. *Journal of Biomechanics*, 24(12):1151–1163.

Viceconti, M., Astolfi, L., Leardini, A., Imboden, S., Petrone, M., Quadrani, P., Taddei, F., Testi, D., and Zannoni, C. (2004). The multimod application framework. *Information Visualisation, International Conference on*, 0:15–20.

Von Funck, W., Theisel, H., and Seidel, H. P. (2008). Volume-preserving mesh skinning. In *Vision Modeling and Visualization*, pages 409–414. IOS Press.

VPHOP (2010). the osteoporotic virtual physiological human, http://vphop.eu.

Xu, W.-W. and Zhou, K. (2009). Gradient domain mesh deformation a survey. *Journal of Computer Science and Technology*, 24(1):6–18.

Zhou, K., Huang, J., Snyder, J., Liu, X., Bao, H., Guo, B., and Shum, H.-Y. (2005). Large mesh deformation using the volumetric graph laplacian. *ACM SIGGRAPH 2005 Papers on SIGGRAPH 05*, 1(212):496.