INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN BIOMEDICAL ENGINEERING
Int. J. Numer. Meth. Biomed. Engng. 2010; **26**:977–998
Published online 10 March 2010 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cnm.1374

A meshless Total Lagrangian explicit dynamics algorithm for surgical simulation

Ashley Horton, Adam Wittek, Grand Roman Joldes and Karol Miller*, †

Intelligent Systems for Medicine Laboratory, School of Mechanical Engineering, The University of Western Australia

SUMMARY

A method is presented for computing deformation of very soft tissue. The method is motivated by the need for simple, automatic model creation for real-time simulation. The method is meshless in the sense that deformation is calculated at nodes that are not part of an element mesh. Node placement is almost arbitrary. Fully geometrically nonlinear Total Lagrangian formulation is used. Geometric integration is performed over a regular background grid that does not conform to the simulation geometry. Explicit time integration is used via the central difference method. As an example the simple but fully nonlinear Neo-Hookean material model is employed. The results are compared with a finite element simulation to verify the usefulness of the method. Copyright © 2010 John Wiley & Sons, Ltd.

Received 27 November 2008; Revised 5 October 2009; Accepted 5 October 2009

KEY WORDS: meshless methods; Total Lagrangian formulation; explicit dynamics; biomechanics

1. INTRODUCTION

Calculations of soft tissue deformation for patient-specific surgical simulations are typically based on Finite Element Analysis (FEA) [1–8]. The results from these finite element experiments are promising, demonstrating that a high level of precision can be achieved in real-time simulations of surgical procedures using nonlinear (both geometric and material) biomechanical models. The accuracy of the finite element calculation relies heavily on the element mesh that discretizes the geometry; hence, it is desirable to use only good-quality hexahedral elements. When the geometry is highly irregular, an experienced analyst is required to manually create such a mesh, which consumes valuable human time. The good hexahedral mesh used in [6] took more than 2 months to be generated by an experienced analyst. This is a major bottleneck in the efficient generation of patient-specific models to be used in the real-time simulation of surgical procedures.

One solution is to use a numerical method that does not require such strict spatial discretization. This paper proposes a meshless method using an unstructured cloud of nodes to discretize the geometry instead of elements. Placement of these nodes is automatic since their arrangement is almost arbitrary. A further advantage of using a meshless method is the ability to deal with extremely large deformations and boundary changes [9–12] that occur during neurosurgical procedures such as retractions, cuts and tissue removal.

*Correspondence to: Karol Miller, Intelligent Systems for Medicine Laboratory, School of Mechanical Engineering, The University of Western Australia.
†E-mail: kmiller@mech.uwa.edu.au

Contract/grant sponsor: Australian Research Council; contract/grant number: DP0664534

In [13], a craniotomy-induced brain shift was simulated with both an Element-Free Galerkin method [14] and a finite element method in LS DYNA. Although slightly slower, the Element-Free Galerkin method gave very similar results to the finite element method and can certainly be considered for use in future simulations. The major limitation of LS DYNA's implementation of the Element-Free Galerkin method is that it requires a complete mesh of hexahedral elements that conform closely to the geometry. By employing this mesh of hexahedral elements, LS DYNA ensures that nodes are evenly distributed through the problem domain and thus reduces the possibility of near singular shape functions (see Section 2.4) and disconnected integration points (see Section 3.2). Background integration is also performed over these elements, hence the volume of integration matches the actual model volume almost perfectly. However, the requirement of a structured hexahedral mesh removes the method's ability to deal easily with irregular geometries.

An algorithm that does not require a geometry conforming hexahedral element mesh is a necessary step toward using meshless methods for fast, efficient simulation of surgical procedures. For this reason, this study proposes an algorithm that integrates over a regular background grid that does not conform to the simulation geometry. Independent of the background integration grid are the nodes where displacements are calculated. To allow for complicated boundaries, the method can accept an almost[‡] arbitrary placement of nodes throughout the simulation geometry. Both the integration grid and the node placement for the algorithm can be created automatically for any geometry that distinguishes it from the hexahedral-dependant Element-Free Galerkin offered in LS DYNA.

Any algorithm used in patient-specific surgical simulations must be capable of producing dynamic results in real-time. Most commercial dynamic finite element solvers use Updated Lagrangian formulation, whereas the proposed algorithm uses Total Lagrangian formulation. The difference between these methods is that in Updated Lagrangian formulation variables are calculated with reference to the previous calculated configuration, as opposed to Total Lagrangian formulation, in which they are calculated with reference to the initial configuration [2].

As in [7], the proposed method precomputes the constant strain–displacement matrices for each integration cell and uses the deformation gradient to calculate the full matrix at each timestep. The proposed algorithm uses explicit time integration based on the central difference method. Unlike implicit time integration, this does not require solving systems of equations at every timestep.

Owing to its construction, the proposed method is called the Meshless Total Lagrangian Explicit Dynamics method (MTLED).

This paper is divided into three main sections, namely Algorithm, Algorithm Parameters and Algorithm Verification. The Algorithm section gives a broad outline of the proposed method but leaves discussion of the critical parameters to the following section. Finally, the proposed numerical method is verified against an existing, established finite element method (Abaqus). The three sections are presented in this form for clarity, although in reality the method was developed alongside the parameter choice. The algorithm was constantly verified right from the start and relevant insights were taken back to develop the algorithm and parameters. For this reason, the discussion in the parameter section often refers to geometries and experimental setups that are not fully described until the verification section.

2. ALGORITHM

2.1. *Algorithm overview*

The following is an overview of the MTLED algorithm. Notation is based on that used in [2]. Detailed explanations of each part are given in the following sections.

[‡]Totally arbitrary placement will never be possible. An extreme example of this is to imagine all nodes being placed at the same location.

Preprocessing

1. Load simulation geometry Ω in the form of two lists:
   - Node locations.
   - Integration point locations.
2. Load boundary conditions.
3. Loop through list of integration point locations. For each integration point:
   - Identify n local nodes.
   - Create and store the $3\times n$ matrix $D\Phi(\mathbf{x})$ of Moving Least-squares shape function derivatives

$$D\Phi_{k,i}(\mathbf{x})=\frac{\partial\phi_i(\mathbf{x})}{\partial x_k},\quad k=1,2,3,\quad i=1,2,\ldots,n \tag{1}$$

4. Loop through nodes and associate with each a suitable mass.
5. Initialize global nodal displacements ${}^{-\Delta t}\mathbf{U}$ and ${}^{0}\mathbf{U}$.

Solving
In every timestep t:

1. Loop through integration points[§].
   - From precomputed list, find n local nodes and associated shape function derivatives $D\Phi(\mathbf{x})$ for the given integration point $\mathbf{x}$.
   - Find $n\times 3$ local nodal deformation matrix ${}^{t}u$.
   - Calculate deformation gradient ${}^{t}_{0}X$.
   - Calculate strain–displacement matrix ${}^{t}_{0}B_L$.
   - Calculate second Piola–Kirchoff stress vector ${}^{t}_{0}\hat{S}$ (using constitutive law).
   - Calculate and store local nodal reaction forces

$${}^{t}_{0}f=\int_{V^0}{}^{t}_{0}B^{\mathrm{T}}_{L}{}^{t}_{0}\hat{S}\,\mathrm{d}V^0 \tag{2}$$

2. Combine all local nodal reaction forces to create global nodal reaction forces vector ${}^{t}_{0}\mathbf{F}$.
3. Calculate global nodal displacements at time $t+\Delta t$ using central difference method

$${}^{t+\Delta t}\mathbf{U}=-\Delta t^2\mathbf{M}^{-1}({}^{t}_{0}\mathbf{F}-{}^{t}\mathbf{R})+2{}^{t}\mathbf{U}-{}^{t-\Delta t}\mathbf{U} \tag{3}$$

   where $\mathbf{M}$ is the diagonal mass matrix and ${}^{t}\mathbf{R}$ is the load applied at time t.

2.2. Geometry discretization

Any given simulation geometry $\Omega\in\mathbb{R}^3$ is discretized by nodes (where mass exists and forces and displacements are calculated) and integration points (where stresses and strains are calculated). Both nodes and integration points exist as particles in the geometry with no connection to each other before support domains and shape functions are created.

A major advantage of meshless methods over hexahedral-based finite element methods is the freedom of node placement. This freedom has some limitations that are discussed in Section 2.4. In general, what is needed is a roughly even density of nodes throughout the domain.

When calculating nodal reaction forces in a small region $V\in\Omega$, numerical integration is used. The general form for numerical integration of a function f over a region V is

$$\int f(x)\,\mathrm{d}V\approx\sum f(x_i)w_i \tag{4}$$

[§]Technically, we should be looping through integration regions (background cells). We use single-point integration, hence this is equivalent.

which leaves the questions of how to partition Ω into many smaller regions such as V, where to place $x_i \in V$ and what weights w_i to use. Available frameworks include

Background finite element mesh. A mesh is created conforming to the nodes and standard finite element integration methods are used.

Nodal integration. Some or all nodes are used as single sampling points and the weights are set as the volume associated with each node. Given the integration points, Ω is partitioned via the Voronoi decomposition.

Background grid. A regular grid of cells is imposed over the geometry and integration is performed in each cell using standard techniques for a regular 3D region.

Creating a background mesh removes most of the flexibility of the meshless method. A hexahedral mesh would likely require manual construction for complicated geometries making a tetrahedral mesh a better option. The tetrahedral integration mesh can be constructed to conform or not conform to the existing nodes. Conforming to the nodes restricts the possibility of varying integration densities independently to the nodes. A non-conforming tetrahedral mesh has more promise because the simulation volume can then be discretized automatically and would be accurately modeled.

Nodal integration is said to be fast and efficient [15], but this is claimed in comparison to background meshes and background grids, which use several gauss points per region.[¶] The speed of nodal integration is balanced by its instability [16].

In this study a regular background grid is used with single-point integration for each cell. This is fast in theory because of the low number of integration points and in practice because the simplicity lends itself to efficient coding. The accuracy and stability of this method is supported by the results given in Section 4.5.

An important issue that arises from the use of a regular background mesh is the number of integration points that should be used. This choice is significant for both speed and accuracy and will be dealt with in Section 3.2.

2.3. Support domains

Consider an arrangement of nodes in the problem domain Ω. At each of these nodes, we attach a field variable that in the case of mechanical deformation represents the displacement the node undergoes. To find the displacement of a point $\mathbf{x} \in \Omega$ that is not a node (for example, an integration point), we must consider the field variables at nearby nodes and create some approximate interpolation.

Before considering the interpolation (which will be performed by Moving Least-squares shape functions) we need an idea of what the local nodes for an integration point are. The 'Support Domain' of a point $\mathbf{x} \in \Omega$ is some bounded region $S \subset \Omega$ that contains both $\mathbf{x}$ and at least one node.

The construction of the support domains is of critical importance since these are what bind nearby but otherwise disjoint nodes together, as well as distant nodes through overlapping support domains. The practical side of this is discussed in Section 3.2.

In $\mathbb{R}^3$ the simplest and the most common form of support domain is a sphere centred on $\mathbf{x}$ with radius r. If $\mathbf{x}_i$ is any node in Ω, then

$$\mathbf{x}_i \in S \Longleftrightarrow \|\mathbf{x} - \mathbf{x}_i\| \leqslant r \tag{5}$$

Of course the norm need not be Euclidean if spherical support domains are not desired. The value of r is based on factors such as the size of Ω, the density of nodes and the required accuracy. It should also be noted that for non-convex Ω, an extra condition must be placed on $\mathbf{x}_i$ to be

[¶]Nodal integration by nature involves only single-point integration.

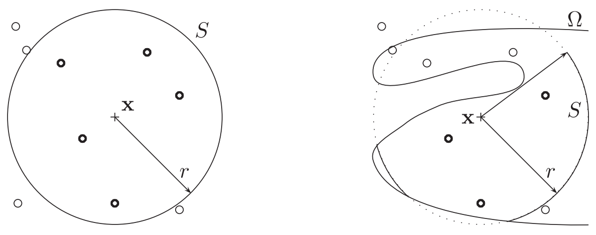

Figure 1. Support domains S with included (black) and excluded (white) nodes. Note the convex geometry of Ω on the right and the reduced support domain.

included in the support domain S.

$$\mathbf{x}_i \in S \Rightarrow (\lambda\mathbf{x}+(1-\lambda)\mathbf{x}_i) \in \Omega \quad \forall \lambda \in [0,1] \tag{6}$$

This excludes nodes that are 'around the corner' (see [14]) to avoid unrealistic direct connections between points in the geometry (Figure 1).

Having defined a support domain S and identified the nodes included in S for an arbitrary point $\mathbf{x}$, a distinction is made between nodes that are very close to $\mathbf{x}$ and those that are only just inside S. This is done by weighting according to an appropriate function of their straightline distance from $\mathbf{x}$.

In this study, the quartic spline weight function is used.

$$W(d_i) = 1-6d_i^2+8d_i^3-3d_i^4, \quad d_i = \frac{\|\mathbf{x}-\mathbf{x}_i\|}{r} \tag{7}$$

This weight function has been used for its simplicity and for the fact that it has been used with success elsewhere. Of the various weight functions discussed in [14, 15], negligible differences in the results were observed.[‖] For further discussion of support domains and weight functions, see [14, 15].

In contrast to the traditional support domain described above, the proposed method uses a fixed number n (user defined) of nodes to be included in each support domain and hence the size (r) of our support domains becomes dependant on the node placement. The preprocessing stage finds the closest (in the Euclidean norm sense) n nodes to each integration point rather than finding all the nodes in a predetermined local volume. During the simulation, the code loops through the nodes in each support domain many times; hence; having a constant n makes this method more memory and processor operation efficient. To assign weights, r is artificially defined as

$$r = \alpha\left(\max_i d_i\right), \quad \alpha > 1 \tag{8}$$

for each support domain and substituted into Equation (7). Changing α scales the influence of nodes based on their location in the support domain, as when $\alpha \rightarrow \infty$ the weights of all nodes become equal.

2.4. Moving Least-squares shape functions

In this study, Moving Least-Squares shape functions are used for their simplicity and robustness. These shape functions were initially developed by Lancaster and Salkauskas [17] and used in meshless methods in the Diffuse Element Method of [18].

[‖]In this study, very few nodes are used per integration point; hence, all nodes are of similar importance to a given integration point and the weight function does not distinguish them greatly. Hence, the insignificance of the choice of weight function is expected.

Let $u(\mathbf{x})$ define the field variable of the point $\mathbf{x}\in\Omega$. The function $u(\mathbf{x})$ is approximated with $u^h(\mathbf{x})$:

$$u^h(\mathbf{x}) = \sum_j^m p_j(\mathbf{x})a_j(\mathbf{x}) \tag{9}$$

$$= \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a}(\mathbf{x}) \tag{10}$$

where m is the number of basis functions (user defined) and $\mathbf{a}(\mathbf{x})$ is an m-vector of coefficients that must be calculated. Low-order monomials are used as basis functions, but it is possible to use more exotic functions in the basis to deal with singularities that may arise in certain problems. This idea is not unlike the extended and the enriched finite element methods of [19, 20].

In $\mathbb{R}^3$, the m-vector of monomial basis functions is

$$\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = (1\ \ x\ \ y\ \ z\ \ xy\ \ xz\ \ yz\ \ x^2\ \ y^2\ \ z^2\ \ x^2y\ \ x^2z\ \ xyz\ \ \ldots\ \ x^k\ \ y^k\ \ z^k) \tag{11}$$

for some $k\in\mathbb{Z}^+$, which represents the highest order of our monomial basis functions.

Suppose for a particular point $\mathbf{x}\in\Omega$, the support domain S of $\mathbf{x}$ contains n nodes $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. At each of these nodes the field function has values of $u_1, u_2, \ldots, u_n$, respectively, which are approximated by using Equation (10) (after calculating $\mathbf{a}(\mathbf{x})$).

$$u^h(\mathbf{x}_i) = \mathbf{p}^{\mathrm{T}}(\mathbf{x}_i)\mathbf{a}(\mathbf{x}), \quad i = 1, 2, 3, \ldots, n \tag{12}$$

Next, a functional J is created from the weighted residuals and minimized.

$$J = \sum_i^n W(d_i)(u^h(\mathbf{x}_i) - u(\mathbf{x}_i))^2 \tag{13}$$

$$= \sum_i^n W(d_i)(\mathbf{p}^{\mathrm{T}}(\mathbf{x}_i)\mathbf{a}(\mathbf{x}) - u_i)^2 \tag{14}$$

$$d_i = \|\mathbf{x} - \mathbf{x}_i\| \tag{15}$$

For an arbitrarily chosen $\mathbf{x}\in\Omega$, $\mathbf{a}(\mathbf{x})$ is chosen to minimize J by considering

$$\frac{\partial J}{\partial \mathbf{a}} = 0 \tag{16}$$

which is equivalent to the matrix system

$$\mathbf{A}(\mathbf{x})\mathbf{a}(\mathbf{x}) = \mathbf{B}(\mathbf{x})\mathbf{U}_S \tag{17}$$

$\mathbf{A}(\mathbf{x})$ is an $m\times m$ matrix, known as the weighted moment matrix, and is defined as

$$\mathbf{A}(\mathbf{x}) = \sum_i^n W(d_i)\mathbf{p}(\mathbf{x}_i)\mathbf{p}^{\mathrm{T}}(\mathbf{x}_i) \tag{18}$$

$\mathbf{B}(\mathbf{x})$ is an $m\times n$ matrix defined as

$$\mathbf{B}(\mathbf{x}) = [\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n] \tag{19}$$

$$\mathbf{B}_i = W(d_i)\mathbf{p}(\mathbf{x}_i) \tag{20}$$

Finally, $\mathbf{U}_S$ is an n-vector containing the value of the field variables at each node in the support domain.

$$\mathbf{U}_S = (u_1 u_2, \ldots, u_n)^{\mathrm{T}} \tag{21}$$

n and m are chosen to make it very improbable that $\mathbf{A}(\mathbf{x})$ will be singular. In the event that $\mathbf{A}(\mathbf{x})$ is still singular, there are adjustments to make it invertible (see [11]). For now it is assumed that $\mathbf{A}(\mathbf{x})$ is invertible.

Equation (17) is used to find $\mathbf{a}(\mathbf{x})$:

$$\mathbf{a}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x})\mathbf{U}_S \tag{22}$$

which is substituted back into (12) to obtain

$$u^h(\mathbf{x}) = \sum_i^n \sum_j^m \mathbf{p}_j(\mathbf{x})(\mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x}))_{j,i} u_i \tag{23}$$

$$= \sum_i^n \phi_i(\mathbf{x}) u_i \tag{24}$$

where $\phi_i(\mathbf{x})$ is the shape function at the ith node in the support domain and is defined as

$$\phi_i(\mathbf{x}) = \sum_j^m \mathbf{p}_j(\mathbf{x})(\mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x}))_{j,i} \tag{25}$$

The n-vector of shape functions is $\Phi(\mathbf{x})$ and we can write the approximate function of the field variable as

$$u^h(\mathbf{x}) = \Phi(\mathbf{x})\mathbf{U}_S \tag{26}$$

This concludes the construction of the Moving Least-Squares shape functions for given n nodes close to a point $\mathbf{x} \in \Omega$ of interest. The algorithm presented in this study only requires the $3n$ first partial spatial derivatives ($\partial\phi_i(\mathbf{x})/\partial x_k$ for $k = 1, 2, 3$) of these shape functions, which are calculated entirely in the preprocessing stage.

These polynomials are of first order and the accuracy of such approximations, including their ability to reproduce a given function, is explored further in Finite Element texts such as [2].

2.5. *Mass allocation*

All mass in the simulation is located at the nodes. Each integration cell is allocated a mass based on its volume and density. This mass is split evenly to the n nodes in the support domain of that cell. Many nodes will thus have different masses proportional to the number of support domains in which they are included. This is a good method of distributing mass because nodes in more support domains will receive more forces. For good results, every node should be included in at least two support domains and preferably in three to four with integration points roughly surrounding the node.

The result of a node being included too in few support domains is low mass and unbalanced forces. This leads to high acceleration and an unstable simulation thanks to the explicit time integration. Even worse is the case of a node that escapes all support domains. No force will be applied to that node, but its massless nature means that it must be removed entirely or the diagonal mass matrix will be singular.

The simplest method to avoid nodes with abnormally low or zero mass is to increase the density of the background integration grid. This will reduce the volume and hence the mass of each cell but creates a more even distribution of mass across the entire simulation volume. The obvious cost is an increased computation time. The density of integration points is discussed in detail in Section 3.2.

2.6. *Force calculation*

The proposed algorithm uses a Total Lagrangian formulation for calculating forces from stresses and displacements. This differs from the traditional method used in most commercial finite element packages that uses Updated Lagrangian. The difference being that Updated Lagrangian calculates the variables with reference to the previous timestep, whereas Total Lagrangian calculates with reference to the initial configuration. Total Lagrangian formulation requires much more computer memory than Updated since all the shape function derivatives for the entire domain must be calculated, stored and quickly accessed many times during the simulation. The choice made by commercial packages to use Updated Lagrangian dates back to a time when internal memory was much more expensive and slower than it is today. Today memory is fast and cheap enough to make Total Lagrangian formulation feasible and to reap the benefits of not having to solve any systems of equations at every timestep.

From [2] we have the Total Lagrangian formulation

$$ {}_0^tF = \int_{{}^0V} {}_0^tB_{L0}^{\mathrm{T}}\,{}_0^t\hat{S}\,\mathrm{d}^0V \tag{27}$$

which is integrated numerically.

The full strain–displacement matrix ${}_0^tB_L$ has the following construction:

$$ {}_0^tB_L = [{}_0^tB_L^{(1)}, {}_0^tB_L^{(2)}, \ldots, {}_0^tB_L^{(n)}] \tag{28}$$

$$ {}_0^tB_L^{(i)} = {}_0^tB_{L0}{}^{(i)}\,{}_0^tX^{\mathrm{T}} \tag{29}$$

$$ {}_0^tB_{L0}^{(i)} = \begin{pmatrix} \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_1} & 0 & 0 \\ 0 & \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_2} & 0 \\ 0 & 0 & \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_3} \\ \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_2} & \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_1} & 0 \\ 0 & \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_3} & \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_2} \\ \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_3} & 0 & \dfrac{\partial\phi_i(\mathbf{x})}{\partial x_1} \end{pmatrix} \tag{30}$$

where every element of ${}_0^tB_{L0}^{(i)}$ is taken from the precomputed $D\Phi(\mathbf{x})$. This update is fast and efficient since all the shape function calculations are done in the preprocessing stage. In contrast, an Updated Lagrangian formulation would involve recalculating the shape functions and their derivatives at every timestep and for every integration point.

2.7. *Explicit time integration*

The $3n$ nodal forces calculated at each integration point are combined to form the global force vector ${}_0^t\mathbf{F}$. These forces are the only data that are stored at each step of the integration point loop.

Use

$$ \mathbf{M}\,{}^t\ddot{\mathbf{U}} = {}^t\mathbf{R} - {}_0^t\mathbf{F} \tag{31}$$

where the forces on the right-hand side are the difference between the applied forces and the reaction forces calculated in Section 2.6. Mass is constant; hence, the finite difference method is applied to acceleration to find

$$ {}^t\ddot{\mathbf{U}} \approx \frac{1}{\Delta t^2}({}^{t-\Delta t}\mathbf{U} - 2\,{}^t\mathbf{U} + {}^{t+\Delta t}\mathbf{U}) \tag{32}$$

Putting (32) into (31) gives

$$ {}^{t+\Delta t}\mathbf{U} = \Delta t^2\mathbf{M}^{-1}({}^t\mathbf{R} - {}_0^t\mathbf{F}) + 2\,{}^t\mathbf{U} - {}^{t-\Delta t}\mathbf{U} \tag{33}$$

This concludes one timestep. If the simulation involves any enforced displacements, they are enforced here by adjusting ${}^{t+\Delta t}\mathbf{U}$ appropriately.

The explicit time integration used here is a second-order approximation. The accuracy and stability of this integration is explored in [21] for the Finite Element method and the theory is relevant here.

3. ALGORITHM PARAMETERS

3.1. Shape functions

When choosing shape function parameters such as m, n and α (see Sections 2.3 and 2.4), the only concern is accuracy and speed in the solving part of the algorithm outlined in Section 2.1. Within reasonable limits, the speed of the preprocessing stage (where the shape functions and their derivatives are calculated) is not of great interest.**

Although the proposed algorithm involves many stages and calculations, by counting the number of operations and the size of significant loops in the code, a relationship between certain parameters and computational speed can be obtained. The time taken to perform a single integration step for the numerical experiments is approximately

$$C(n+D) \tag{34}$$

where C is a constant that depends on many factors (including the number of spatial dimensions, processor speed, reading/writing data speeds), and D depends upon the material formulation.

The above function simplifies reality, but it shows that the time for a single integration step is an affine function of the number of nodes in the support domain. The simulations' run in this study involves a simple material formulation and it is found that $D \approx 1.5$. Exploration of other more complicated hyperelastic material models (see [22]) always reveals that $D < 2.5$. Clearly, the choice of n strongly dictates the speed of geometric integration. The cost of very low n is reduced interactions between nodes. Consider the extreme case of $n=1$ in which every node in the entire simulation acts independently of the others. Clearly n must be large enough to allow nodes to exist within multiple support domains. As mentioned in Section 2.5, any node that exists in only one support domain will experience unbalanced forces leading to the termination of the simulation.

In fact the choice of n is based on m, the number of monomial basis functions used. In order to create Moving Least-Squares shape functions, it is essential that $n \geqslant m$, but in practice $n \approx 2m$ is a good ratio that gives a statistically small chance of poor-quality shape functions. This has been learnt by generating many geometries, discretising with nodes and comparing ratios of n and m. To choose $n > 2m$ reduces the quality of the shape functions as the polynomials attempt to approximate functions of a much higher degree (through more points) than they are capable of.

When choosing m, the options are limited if shape functions must interpolate similarly in each dimension. The possible lengths of the vector of monomial basis functions (up to quadratic order) are 1, 4, 7 and 10.

$$\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = (1|x \;\; y \;\; z|xy \;\; xz \;\; yz|x^2 \;\; y^2 \;\; z^2) \tag{35}$$

Owing to finite computation power and the necessity of overlapping support domains, the trade-off is between the size of m and the total number of integration points. This relationship is further studied in Section 3.2. The proposed algorithm uses small m and more integration cells with single-point integration, which is better suited to lower-order interpolations. In theory more integration points also allows for more parallelization (see Section 4.9), but in practice the gains from this are not significant unless the number of processors used is of the same order as the number of

**The practical justification for this is that minutes in the initialization stage are of significantly less value than seconds during a surgical procedure.

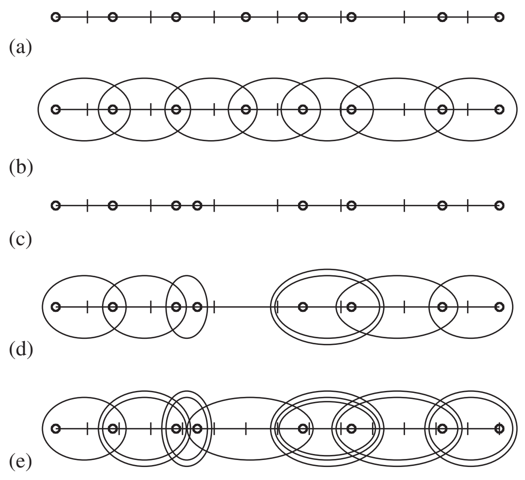

Figure 2. (a), evenly spaced nodes. (b), good support domains for evenly spaced nodes. (c), slightly irregularly spaced nodes. (d), bad support domains for irregularly spaced nodes. (e), better support domains with more integration points.

integration points.[††] If $m=4$ then $n=8$ and this is ample for nodes to transfer deformation and forces between support domains.

In this study, a high value of $\alpha=10$ is used to give all nodes an almost identical weighting within a support domain. This is appropriate since single-point integration is used and there are relatively few nodes per support domain. Setting α close to 1 effectively deletes the influence of the furthest node from the integration point along with any other nodes of similar distance. No benefit is gained by removing outer nodes in this way since n remains constant and therefore the removed nodes will still be included in all calculations throughout the simulation. Furthermore, the relationships between support domains are weakened since the furthest node from an integration point is the one most likely to be included in multiple neighbouring support domains.

3.2. Number of integration points

Consider the $\mathbb{R}^1$ geometry in Figure 2(a) that consists of nodes (marked as dots) and integration points (marked as crosses) on a line. Suppose in this $\mathbb{R}^1$ simulation two nodes are required per support domain. As mentioned in Section 2.3 for each integration point, the closest two nodes are found and a loop is drawn around them to represent support domains as seen in Figure 2(b). Observe that the nodes are roughly evenly spaced and produce a nice chain of linking support domains across the entire geometry. Furthermore, one can imagine forces being applied to the ends of the line and the reaction stresses and strains propagating throughout the geometry.

In Figure 2(c), only one node has been moved and yet the resulting support domains in 2(d) clearly form two distinct chains. In this case, forces cannot be transmitted between the two different groups of nodes. Note that, in finding the closest two nodes to an integration point, a support domain does not necessarily need to enclose the integration point when displayed graphically.

This is a good argument to have as regular a node placement as possible, but the proposed method must be able to deal with irregular geometries, which often imply irregular node placements. This problem is not entirely a product of the different approach to creating support domains. Even the traditional method (finding all nodes in a local region of specific size) would have failed in the example given unless the support domains were made so large that shape function quality would surely suffer, resulting in a need for higher-order functions at an enormous computational cost.

The solution presented here is to add more integration points as shown in Figure 2(e). This increases the statistical chance that a small clump of nodes will be split into multiple support

[††]Though this is possible with Graphical Processor Units [23].

domains by placing an integration point closer to the centre of any large gaps between nodes. Adding integration points slows down the computation, but this is the cost of being able to automatically place nodes almost arbitrarily in an irregular patient-specific geometry. To increase efficiency, one can imagine an adaptive method that varies the density of integration points across a problem domain.

As can be seen in Figures 2(d) and (e), the support domains for different integration points are often identical. While this is computationally inefficient, it does not affect accuracy since both mass and reaction forces for nodes in a given support domain are calculated according to the volume associated with the integration point. As more integration points are added, the volume decreases. Of course in $\mathbb{R}^3$, it would be extremely rare for two support domains to entirely overlap.

The example given in Figure 2 is rather simple and in $\mathbb{R}^3$ it is significantly more complicated to find the optimal number of integration points for a given number of nodes. Unlike in the $\mathbb{R}^1$ example, a volume of nodes and integration points is unlikely to be made up of totally disjoint chains of support domains. However, it is common to have many integration points that are not directly connected to some of their six immediate neighbours. The situation is similar to a solid finite element mesh containing holes in the middle. The result is that nodes that are visibly very close to each other can move apart or even through each other with almost no reaction force being produced. Globally, this results in lower reaction forces and distorted geometries.

Several numerical experiments have been performed with a fixed number of nodes in a geometry and a varying number of integration points to find how many integration points were needed for a converged solution. For this purpose, the same cylinder with material properties and boundary conditions as described in Section 4.1 was used though only the compression example was dealt with. The results were compared with the Abaqus hexahedral FEA solution (implicit solver with hybrid formulation linear elements).

In Figures 3, 4 and 5, the effect of increasing the number of integration points is seen for a given number of nodes. Three experiments are shown using 1169, 2391 and 4314 nodes. Note that some figures show slightly less nodes. In the preprocessing stage, the algorithm looks for any nodes that are included in less than two support domains and removes them from the simulation. This ensures that all the nodes have non-zero mass and are acted upon by multiple balancing forces.

In all the three experiments, significantly lower forces are observed when few integration points are used but as the number of integration points increases, so do the magnitudes of the reaction forces. This is because more support domains will overlap better and the holes in the discretization are removed. The existence of these holes has been confirmed by carefully analysing the support domains, but the results are not shown here.

In all experiments, the final figure does not show perfect correlation with the Abaqus solution, but this is a converged result nonetheless. Increasing the ratio of nodes:integration points beyond 1:2 had negligible impact on the final result in all the experiments. For this reason, the proposed method now uses roughly twice as many integration points as nodes for all the experiments. Regardless of the ratio of nodes to integration points used, the fact that the nodes should be placed as regularly as possible will always be true as long as the integration points are being placed regularly. Of course, we do not dwell on this since most of the numerical methods desire regular discretizations and the proposed MTLED algorithm is designed to be less sensitive to node placement than most.

3.3. Timestep size and stability

In the absence of any nonlinear theory regarding stability of meshless explicit methods, the linear finite element theory for determining the maximum stable timestep [21, 24] was used. The reliability of this method is questionable; hence, some experiments were performed and the results are given below.

The dilatational wave speed in the material used in this study is

$$c=\sqrt{\frac{\lambda+2\mu}{\rho}}\approx 7.165 \tag{36}$$

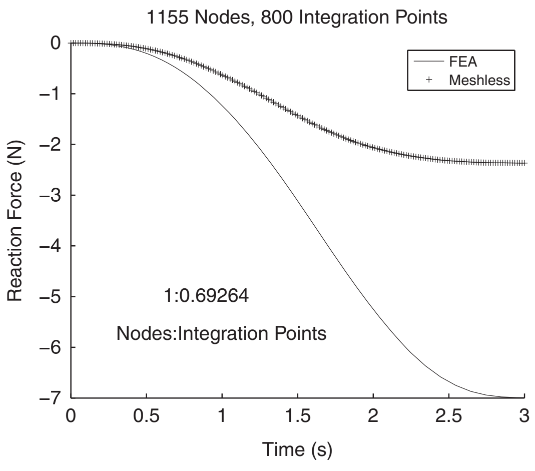
1155 Nodes, 800 Integration Points
FEA
Meshless
Reaction Force (N)
Time (s)
1:0.69264
Nodes:Integration Points
1169 Nodes, 1344 Integration Points
1:1.1497
1164 Nodes, 1820 Integration Points
1:1.5636
1167 Nodes, 2268 Integration Points
1:1.9434

Figure 3. Effect of change in integration-point density. Note that the number of nodes (approximately 1150) is shown to be adequate for good results (in the bottom right plot), but there are sufficient integration points.

where $\lambda = 49.33\,\text{kPa}$, $\mu = 1.007\,\text{kPa}$ and $\rho = 10^3\,\text{kg}\,\text{m}^{-3}$ are the material parameters taken from [25] and discussed in Section 4.2. The critical timestep for a given integration point is

$$\Delta t^{\text{c}} = \frac{2}{\sqrt{g}c} \tag{37}$$

where g is the sum of the squares of all the shape function derivatives for that integration point. Hence, the maximum timestep for a given geometry is obtained by finding the maximum g over all the integration points. For the 4314 node geometry used in this study, $\Delta t^{\text{c}} \approx 0.0005289\,\text{s}$.

To check the validity of this upper bound, several compression experiments were performed with varying timesteps and the calculated reaction forces are shown in Figure 6. The displacement was enforced 10 times faster than in the other experiments performed in this paper in order to represent the worst case. Deformation of this speed is unrealistic in surgical simulation but serves to establish an upper bound for the critical timestep. It was found that all timesteps less than $1.34\Delta t^{\text{c}}$ produce good, stable results but that beyond this threshold the simulation failed prematurely. That is, timesteps up to and including 134% of the critical timestep were stable in this simulation. Figure 6 shows several calculated reaction force curves marked from 1.35 to 1.5 that failed before $t = 0.3\,\text{s}$. From this and other similar experiments, it is deduced that staying close to but below the critical timestep calculated from linear finite element theory is a good way to choose efficient timesteps for the method.

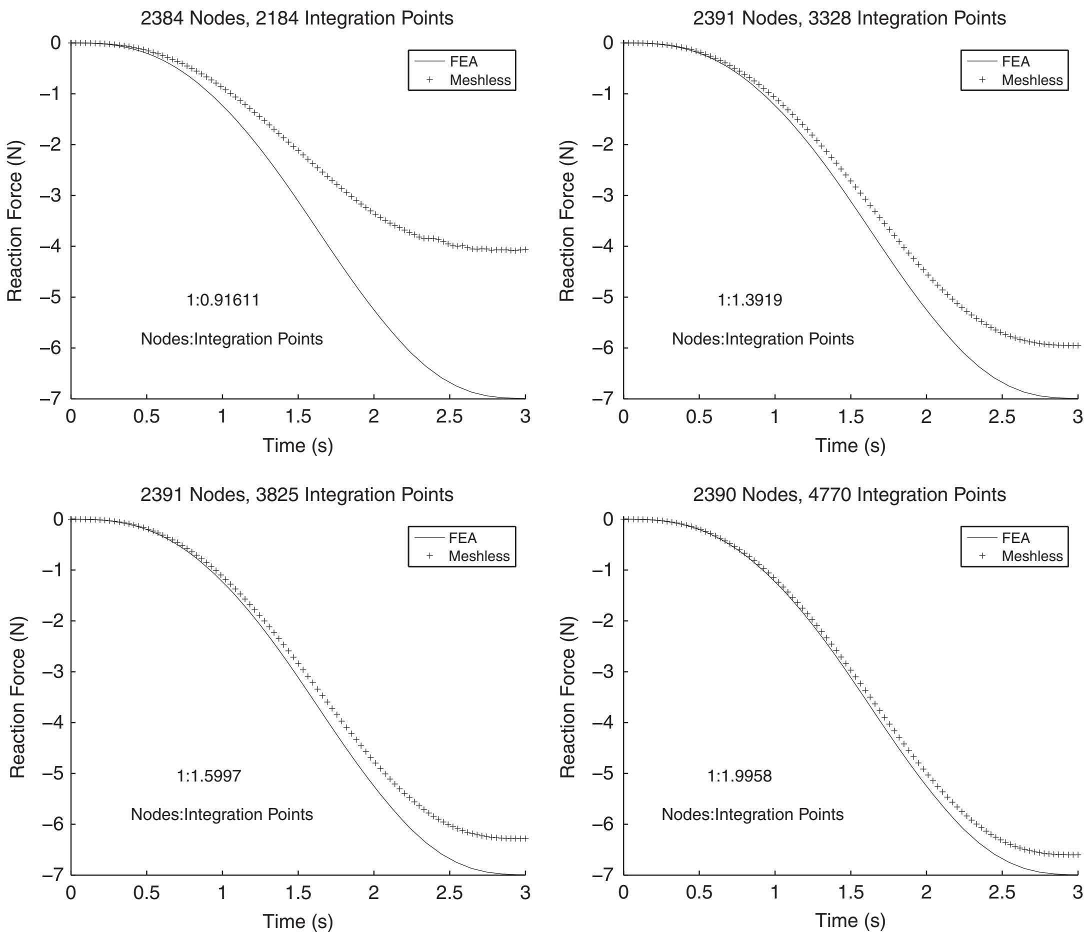

Figure 4. Effect of change in integration-point density. From these plots as well as those in the previous figure, we can tell that the ratio of nodes to integration points is more important that just having a sufficiently large number of integration points.

For more theory regarding timestep stability in Explicit Finite Element Modeling, see [21] and in Total Lagrangian Explicit Dynamics, see [7].

4. ALGORITHM VERIFICATION

To show the accuracy of the proposed method, numerical experiments were performed and the results are compared here with those obtained with an established finite element code. The finite element code used for comparison was Abaqus [26] using the implicit solver with hybrid formulation linear elements. While only selected results are presented in this section, many more experiments that support the claims made here have been performed.

4.1. Geometry

The geometry used for these numerical experiments was a cylinder of height 0.1 m and radius 0.05 m. A cylinder is simple enough to obtain a good finite element mesh for comparison and unrealistic perfect corners are not a major issue (compared with using a cube).

A limitation of many meshless simulations in the literature is the use of very regular node placement. For example, Guangyao and Belytschko [27] performed several experiments with a perfectly regular grid of nodes and integration points were located at the centre of quadrilaterals

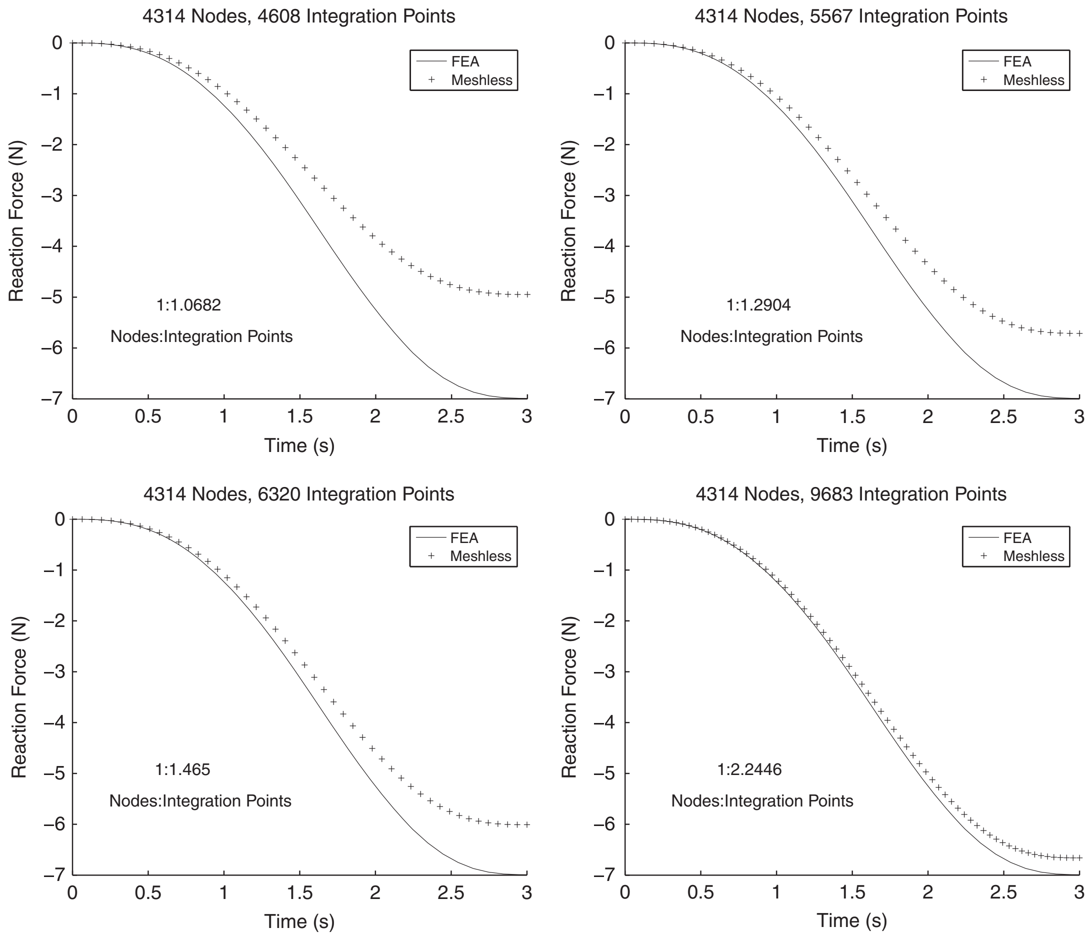

Figure 5. Effect of change in integration-point density. The ratio of two integration points per node is established in this third set of plots.

formed by nodes. This gives excellent results, but such an arrangement of nodes and integration points is equivalent to a good quadrilateral or hexahedral mesh and thus has the same difficulty in the production for irregular patient-specific geometries. Many examples of meshless methods being used successfully with irregular node placements have been limited to $\mathbb{R}^2$ geometries [28, 29] and the methods do not generalize easily to $\mathbb{R}^3$.

In these experiments, both nodes and integration points are placed automatically with no human help. All that is needed to generate the nodes and integration points is a volume definition‡‡ and the desired nodal density.

Figure 7 is an example of unstructured nodes and integration points marked as dots and crosses, respectively. Note how the integration points form a perfectly regular grid that does not conform to the cylindrical geometry.

The finite element mesh used for the Abaqus simulations contained 6000 hexahedral elements and 6741 nodes. This was a well-structured mesh for the purpose of giving good results, against which the proposed MTLED algorithm can be compared.

‡‡While acknowledging that the construction of volumes from medical images is not a trivial task, it is the one that is not dealt with in this study.

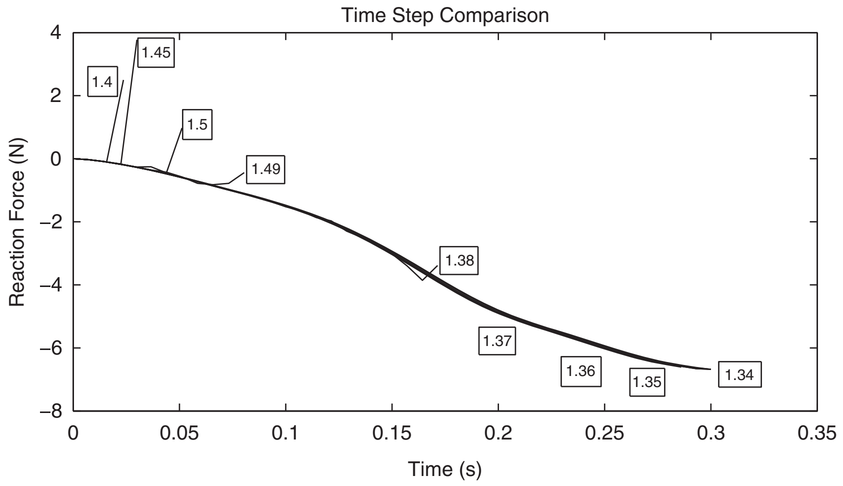

Figure 6. Comparison of reaction forces when the timestep is varied. The numbers marked on the figure show where that multiple of the critical timestep stopped producing machine-sized numerical results (double precision was used in all the calculations). It is easy to see the last data points before failure for the multiples 1.38, 1.4, 1.45, 1.49 and 1.5 of the critical timestep. The multiples of 1.37, 1.36 and 1.35 failed at approximately 0.21, 0.25 and 0.28 s, respectively.

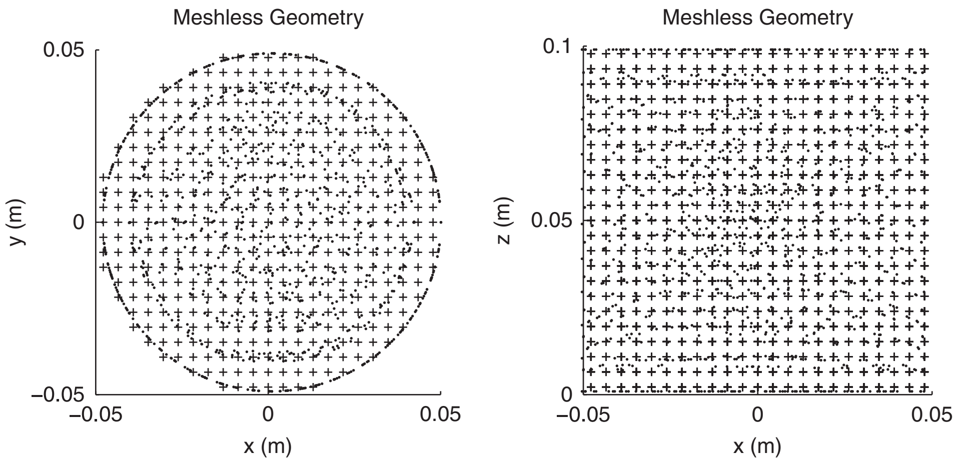

Figure 7. Meshless geometry showing almost arbitrarily placed nodes (.) and non-conforming integration points (+). The orthogonal views are given because any attempt at 3D visualization quickly becomes a very unclear cloud of points. Unlike in finite element meshes, no elements exist for opaque visualization.

4.2. *Material formulation*

For the purpose of algorithm development, it is sufficient to show the proposed algorithm using the Neo-Hookean hyperelastic rubber in these numerical experiments. Neo-Hookean is simple to implement[§§] but still a fully nonlinear material, as is necessary for surgical simulations of soft tissue deformation.

The Neo-Hookean material has the strain–energy density functional

$$W = \frac{\mu}{2}(\bar{I}_1 - 3) + \lambda(J-1)^2 \tag{38}$$

[§§]The focus of this study being on the algorithm rather than on the biomechanical modeling.

Table I. Final displacements (in m) of bottom ($z = 0.0$) and top ($z = 0.1$) surfaces of the cylinder.

	$z=0.0$			$z=0.1$		
Experiment	Δx	Δy	Δz	Δx	Δy	Δz
Compression	0.00	0.00	0.02	0.00	0.00	0.00
Extension	0.00	0.00	−0.02	0.00	0.00	0.00
Shear	0.02	0.00	0.00	0.00	0.00	0.00

where μ, λ are the Lamé parameters, $J = \det({}^t_0X)$ and $\bar{I}_1$ is the first strain invariant of the right Cauchy Green deformation tensor.

$$\bar{I}_1 = J^{-2/3} \text{ trace } ({}^t_0C) \tag{39}$$

$${}^t_0C = {}^t_0X^{\mathrm{T}t}_{\ 0}X \tag{40}$$

$${}^t_0X = I + D\Phi^t(\mathbf{x})U \tag{41}$$

From (38) the second Piola–Kirchoff stress tensor is found.

$${}^t_0S = \lambda J(J-1){}^t_0C^{-1} + \mu J^{-2/3} I \tag{42}$$

and hence form the required vector ${}^t_0\hat{S}$.

The parameters used were $\mu \approx 1.007\,\text{kPa}$ that according to [25] approximate the mechanical response of the brain. With Poisson's ratio of $\nu = 0.49$, this gives $\lambda \approx 49.33\,\text{kPa}$. The mass density in this simulation was $10^3\,\text{kg}\,\text{m}^{-3}$.

4.3. Boundary conditions

Three basic experiments of compression, extension and shear of the cylinder were performed. In all the cases, displacement was enforced on one end in one dimension and the opposite end was held rigidly. Table I shows the actual displacements imposed. All nodes not on one of the ends experience stress-free boundary conditions.

The displacement $u_{\mathbf{x}}$ of a given node $\mathbf{x}$ is enforced over a period of T s up to the maximum $\mathbf{\Delta x}$ according to the smooth function

$$u_{\mathbf{x}}(t) = \mathbf{\Delta x}\left(10 - 15\frac{t}{T} + 6\left(\frac{t}{T}\right)^2\right)\left(\frac{t}{T}\right)^3, \quad 0 < t < T \tag{43}$$

In this study, the total simulation time was $T = 3.0\,\text{s}$.

4.4. Timesteps

A timestep of 0.0005 s was used for all of the experiments presented below. This timestep and the length of the simulations was chosen so that comparison could be made to similar experiments made in [7] but still be slow enough to avoid any inertial artifacts since surgical simulations are quasi-static. The timestep is also stable, see the investigation into stability and the critical timestep in Section 3.3.

4.5. Results

Figure 8 shows the results obtained from 20% compression, extension and shear of the cylinder. Compared with the finite element result, the relative difference in forces observed are no more than 5%. Qualitatively, the forces calculated with the meshless method are seen to be similar to those of the finite element method.

The maximum relative difference in displacement is approximately 3.5% though this can only just be seen in Figure 8 where a meshless node fails to sit exactly on the deformed finite element boundary.

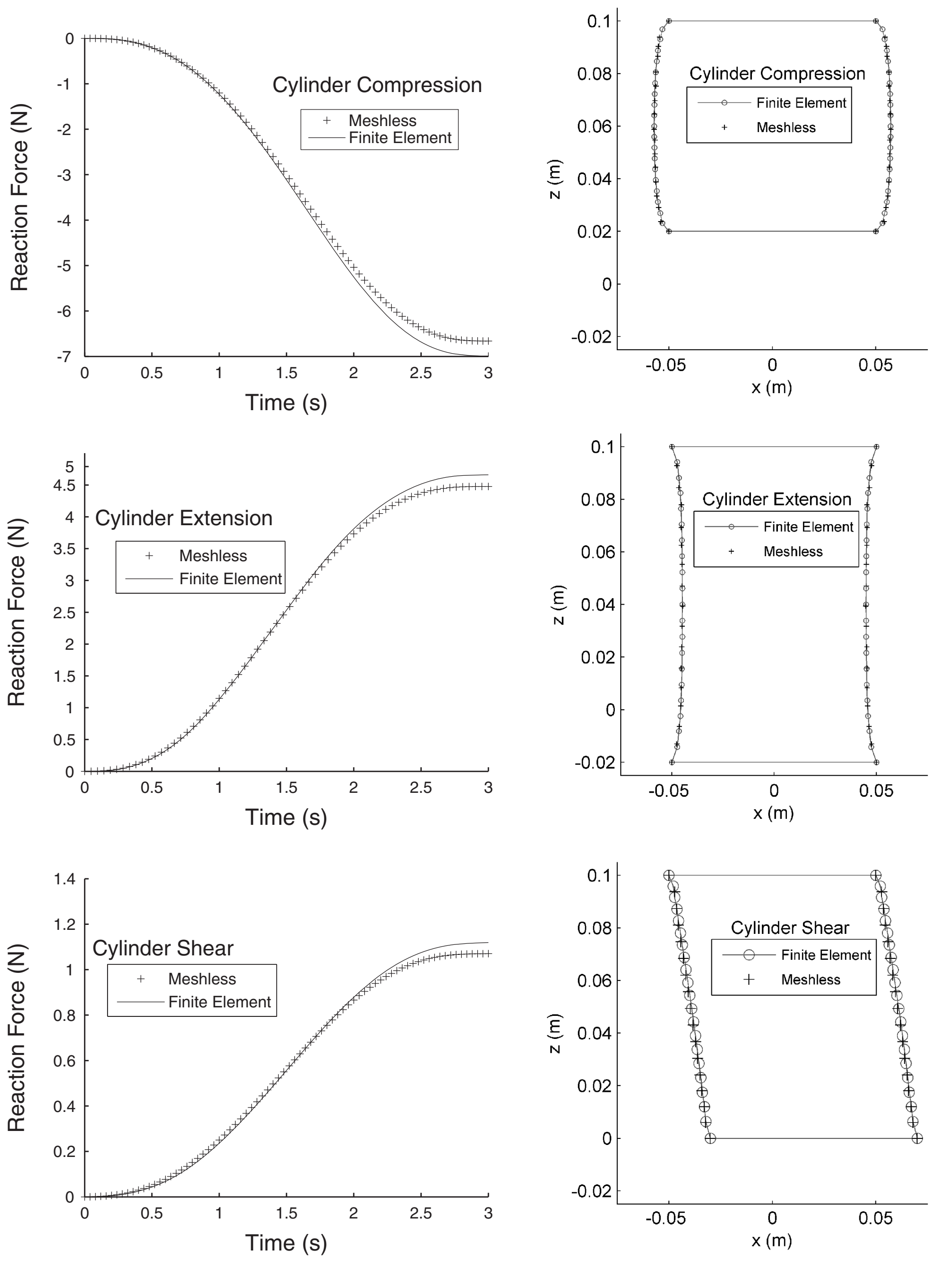

Figure 8. Results from numerical experiments. The left plots show reaction force against time and the right plots show displacement at time $T = 3$ s.

In Figure 9, a force vector has been drawn from each node on the displaced surface and the meshless results are compared with those from Abaqus. The finite element results are intuitively pleasing, whereas the meshless results seem to have large variation. Hence, the proposed meshless method should not be used when reaction forces and displacements of individual nodes are needed, but rather when an overall surface displacement or total reaction force is required. The variation in magnitude of the individual force vectors from the meshless calculation can be accounted for by the fact that some nodes are included in more support domains than others and hence their forces are the result of integration through larger volumes (they also have greater mass). To show that the irregular forces are purely a result of unstructured node placement, Figure 9 also shows the reaction forces calculated by the MTLED method with the same node placements as the finite

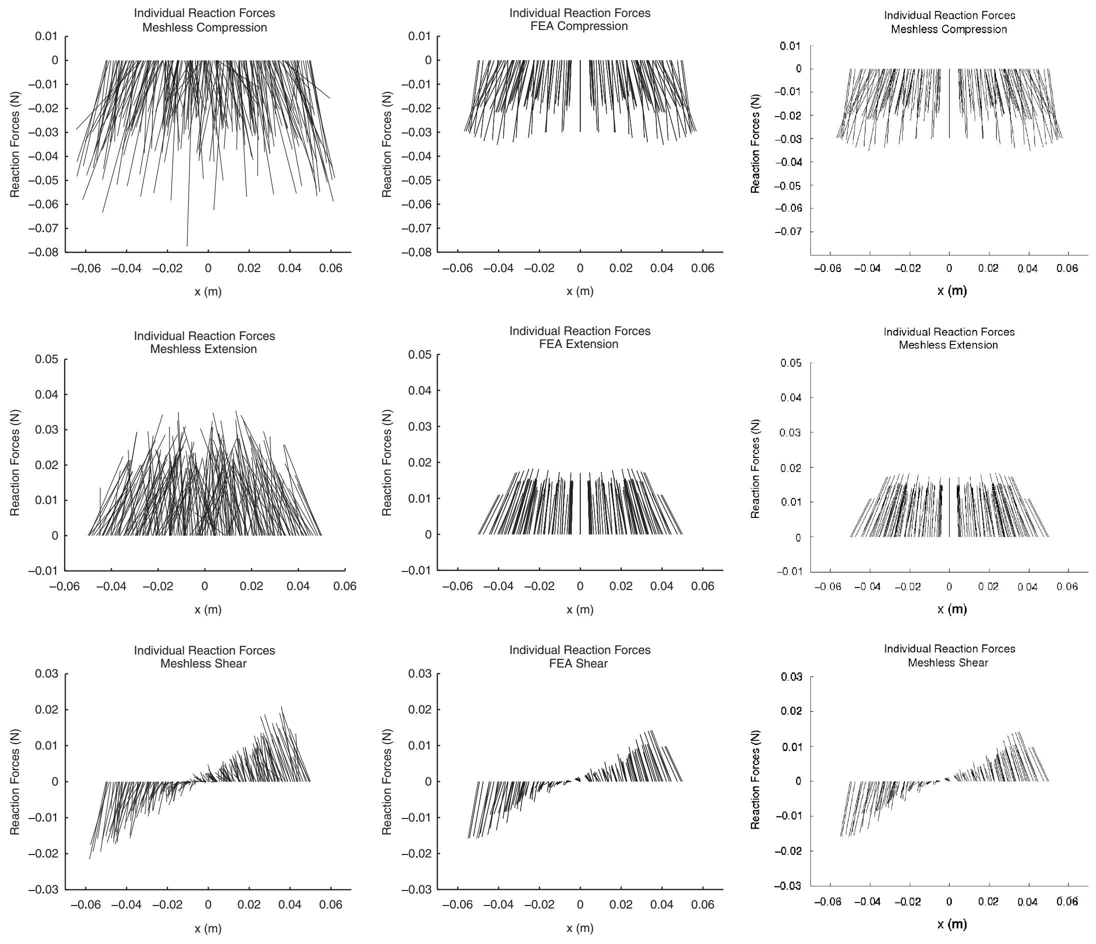

Figure 9. Individual nodal reaction forces from compression (top), extension (middle) and shear (bottom) experiments. MTLED (left), Abaqus Finite Element Analysis (centre) and MTLED with regularly placed nodes (right). Note that these are 3D plots viewed from the side, so only the z-direction forces (vertical in these plots) can be measured accurately on the vertical scale. In the shear case, the vectors have been translated back to their original x-values.

element calculation. Here it is seen that regular forces are obtained that are almost identical to those found with the finite element method.

4.6. Conservation of energy

For the compression experiment, we compared the external work done on the displaced boundary with the internal strain energy. The strain energy was calculated from the strain–energy density functional (38). Internal kinetic energy was calculated but excluded from the plots since it was always at least three orders of magnitude smaller. Figure 10 shows that the proposed method conserves energy well with a maximum relative difference of around 0.01.

4.7. Large deformation

The proposed method is useful for calculating large deformation of soft tissue. To demonstrate this, we simulated compression of the cylinder to 80% of its original height. Validating the results from such experiments is not trivial, since neither visually good results nor algorithm stability implies accuracy. For the sake of validation, we show here the conservation of energy as the cylinder is both compressed and then extended back to its original height. Figure 11 shows that the method produces reasonable results. While several texts, such as [11, 12], also show high compression of

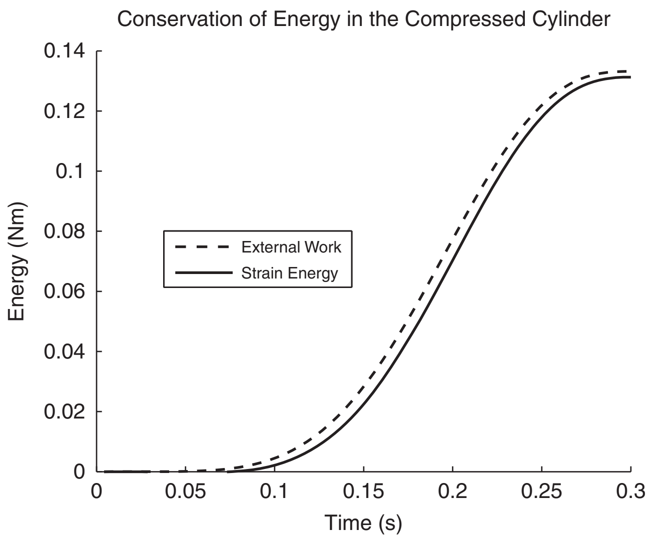

Figure 10. External work and internal strain energy for compression according to the equation in (38).

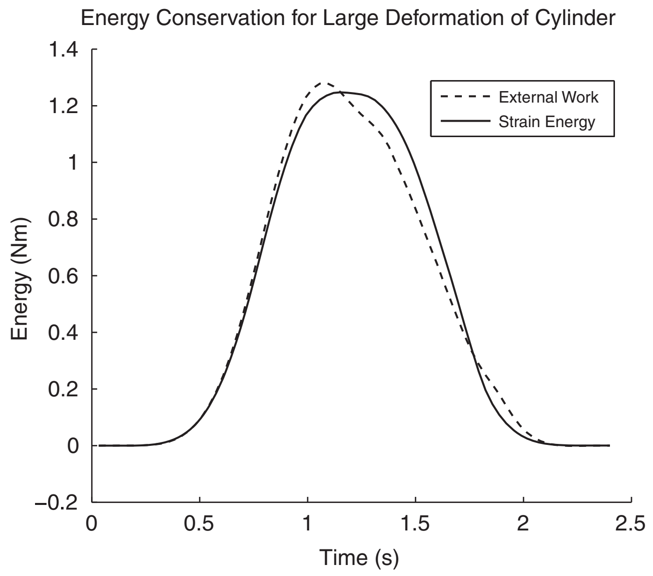

Figure 11. External work and internal strain energy for large compression (and the return to the initial state) according to the equation in (38).

meshless methods and comparisons are made to finite element methods, there are two significant points to make about the result in this study. First, the node placement is almost arbitrary. Greater compression can be achieved with a very regular or structured node placement, but we desire a method that is insensitive to discretization. Second, the material we compress in this experiment is almost incompressible brain tissue. If Poisson's ratio was reduced, then more compression could be achieved.

4.8. Conservation of angular momentum

A final, interesting validation test is to check the conservation of angular momentum. For instance, the Smoothed Particle Hydrodynamics method is known to lose momentum in this case [30, 31]. For this experiment, we used the 4314 node cylinder with 9683 integration points and the same

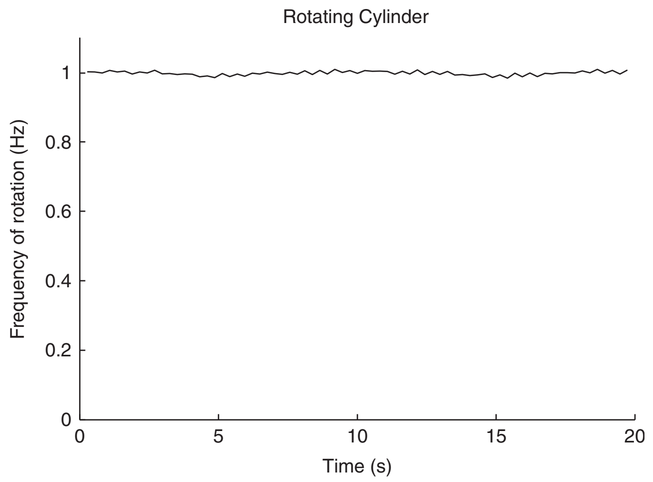

Figure 12. The almost constant rotational frequency shows that the proposed method conserves angular momentum.

material properties as in the other experiments. The cylinder was given an initial frequency of rotation of 1 Hz and allowed to spin for 20 s without any external forces being applied. In Figure 12, we see that the loss of angular momentum is negligible after 20 s.

4.9. *Calculation speed*

Practically speaking, it is useful to compare the speed of the MTLED method with similar finite element methods. Assuming an equal number of nodes and the use of the Total Lagrangian Explicit Dynamics framework, computation time is roughly proportional to the number of integration points. In the finite element method, this is equal to the number of elements if single-point integration is used. The proposed method uses approximately twice as many integration points as a hexahedral based mesh and one third as many as a tetrahedral-based mesh. Hence, the MTLED runs at half the speed of a hexahedral-based finite element simulation but three times faster than a similar tetrahedral-based simulation. This is an impressive speed considering the difficulty of creating good hexahedral-based meshes for patient-specific neurosurgical simulations and the volumetric locking issues associated with tetrahedral elements.

All the calculations were performed on a single Pentium 4 3 GHz machine with 1 GB of RAM under Windows XP Professional. The algorithm was coded in c, compiled with the GNU Compiler Collection and executed in Cygwin. No deliberate optimizations were made for the operating system or processor other than those included in the standard GCC. The structure of the code itself is very simple as it is written for stability and ease of debugging rather than speed. Owing to operating system limitations, the simulation never consumed more than 50% of the computer's CPU.

In spite of the above computation limitations, the experiments still run with reasonable speed. After running 10 simulations for the 4314 node cylinder, the average computation time per timestep was found to be $\approx 14.9\times 10^{-3}$ s. With timesteps around the size of 0.5×10^{-3} s, this simulation is performed approximately 30 times slower than the real-time. By changing the node and hence the integration-point density, the simulation could be speed up or slowed down, that is, the speeds given here are only an example and only apply to this simulation.

While better coding and faster machines will dramatically reduce these times, an improvement of particular interest is parallelization. The explicit time integration at the end of every timestep is many times quicker than the looping through integration points. It is difficult to estimate the difference in computation time for the geometric and time integrations, since the tools used to measure the times leave a sizeable footprint on the data collected. For the experiments performed in this study, the explicit time integration takes between 1 and 2% of a timestep with the geometric integration consuming almost all the rest.

This makes an excellent argument for sending the independent integration point steps to different processors with the time integration being handled by one unit. During the simulation, the only data transferred between secondary processors and the primary unit would be current nodal deformations (to the secondary processors) and nodal reaction forces (back to the primary unit). All material properties and shape function derivatives could be stored at the different processors during the preprocessing stage.

The use of Graphical Processor Units is becoming popular for highly parallel calculations and the results in [23] suggest that a speed up factor of 10 is easily possible for this sort of calculation. Proper optimization of the code could produce the final required speed up factor of 3 to bring this simulation to real-time.

5. CONCLUSIONS

The consistent forces and displacement differences of less than 5% between the finite element results and the meshless simulations shown in Section 4.5 are small and these results show the method to be working and useful. The algorithm will work most effectively in surgical simulation when coupled with the finite element method. In particular, elements should be used for the boundary and as much of the problem geometry as can be meshed with good-quality elements that can be created automatically. For exceptionally complicated boundaries, this may mean only a single layer of hexahedral elements. The meshless algorithm will then be used to fill interior sections where good-quality hexahedral elements cannot be easily created.

The logic here is that the proposed meshless method is accurate in terms of overall reaction forces but not quite as good with individual displacements or forces. A small amount of local discrepancy in the centre of a body is of less importance than on the boundary.

Further work with this method will mainly involve changes that increase speed without loosing accuracy. One anticipated improvement is a new method to automatically place nodes in such a way that allows less integration points. Most of the experiments shown in Section 4.5 use ≈ 2.25 integration points per node but given the shape functions used, there is no reason that this ratio should not be close to 1. This would speed up the simulation by a factor of 2, but for the purpose of this study it is enough to show the method to be stable and accurate for almost arbitrary node placement.

There is also some scope for changing the shape functions, but this will not directly give an increase in computation speeds since only the precomputed derivatives are needed during the main calculation. Computation speed may increase if different shape functions allow bigger support domains and hence less integration points. This will require investigation since bigger support domains make each geometric integration slower even though less integrations need to be performed. The biggest possibility for increasing speed here is through longer timesteps due to smaller shape function derivatives (see Section 3.3).

ACKNOWLEDGEMENTS

The financial support of the Australian Research Council is gratefully acknowledged, Grant No. DP0664534.

REFERENCES

1. Cotin S, Delingette H, Ayache N. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics* 1999; **5**:62–73.
2. Bathe KJ. *Finite Element Procedures*. Prentice-Hall: NJ, 1996.
3. Szekely G, Brechbuhler C, Hutter R, Rhomberg A, Ironmonger N, Schmid P. Modelling of soft tissue deformation for laparoscopic surgery simulation. *Medical Image Analysis* 2000; **4**:57–66.

4. Picinbono G, Delingette H, Ayache N. Non-linear anisotropic elasticity for real-time surgery simulation. *Graphical Models* 2003; **65**:305–321.
5. Luboz V, Chabanas M, Swider P, Payan Y. Orbital and maxillofacial computer aided surgery: patient-specific finite element models to predict surgical outcomes. *Computer Methods in Biomechanics and Biomedical Engineering* 2005; **8**:259–265.
6. Wittek A, Miller K, Kikinis R, Warfield S. Patient-specific model of brain deformation: application to medical image registration. *Journal of Biomechanics* 2007; **40**(4):919–929.
7. Miller K, Joldes G, Lance D, Wittek A. Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. *Communications in Numerical Methods in Engineering* 2007; **23**(2):121–134.
8. Wittek A, Hawkins T, Miller K. On the unimportance of constitutive models in computing brain deformation for image-guided surgery. *Biomechanics in Modeling and Mechanobiology* 2009; **8**(1):77–84.
9. Melenk JM, Babuska I. Partition of unity finite element method: basic theory and applications. *Computer Methods in Applied Mechanics and Engineering* 1996; **139**(1):289–314.
10. Babuska I, Melenk JM. Partition of unity method. *International Journal for Numerical Methods in Engineering* 1997; **40**(4):727–758.
11. Liu GR. *Mesh Free Methods*; *Moving Beyond the Finite Element Method*. CRC Press: Boca Raton, 1996.
12. Li S, Liu WK. *Meshfree Particle Methods*. Springer: Berlin, 2004.
13. Horton A, Wittek A, Miller K. Computer simulation of brain shift using an element free Galerkin method. *Seventh International Symposium on Computer Methods in Biomechanics and Biomedical Engineering*, Antibes, 2006; 906–911.
14. Belytschko T, Lu YY, Gu L. Element-free galerkin methods. *International Journal for Numerical Methods in Engineering* 1994; **37**(2):229–256.
15. Belytschko T, Krongauz Y, Organ D, Fleming M, Krysl P. Meshless methods: an overview and recent developments. *Computer Methods in Applied Mechanics and Engineering* 1996; **139**(1):3–47.
16. Chen JS, Wu CT, Yoon S, You Y. A stabilized conforming nodal integration for Galerkin mesh-free methods. *International Journal for Numerical Methods in Engineering* 2001; **50**:435–466.
17. Lancaster P, Salkauskas K. Surfaces generated by moving least squares methods. *Mathematics of Computation* 1981; **37**:141–158.
18. Nayroles B, Touzot G, Villon P. Generalizing the finite element method: diffuse approximation and diffuse elements. *Computational Mechanics* 1992; **10**(5):307–318.
19. Fleming M, Chu YA, Moran B, Belytschko T. Enriched element-free Galerkin methods for crack tip fields. *International Journal for Numerical Methods in Engineering* 1997; **40**(8):1483–1504.
20. Daux C, Mos N, Dolbow J, Sukumar N, Belytschko T. Arbitrary branched and intersecting cracks with the extended finite element method. *International Journal for Numerical Methods in Engineering* 2000; **48**(12):1741–1760.
21. Hughes TJR. *The Finite Element Method*: *Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall: Englewood Cliffs, 2004.
22. Ogden RW. *Non-Linear Elastic Deformations*. Ellis Horwood Ltd.: Chichester, 1984.
23. Taylor Z, Cheng M, Ourselin S. Real-time nonlinear finite element analysis for surgical simulation using graphics processing units. *Lecture Notes in Computer Science*, vol. 4791. Springer: Berlin, 2007; 701–708.
24. Flanagan DP, Belytschko T. Eigenvalues and stable times steps for the uniform strain hexahedron and quadrilateral. *Journal of Applied Mechanics* 1984; **51**:35–40.
25. Miller K. *Biomechanics of Brain for Computer Integrated Surgery*. Publishing House of the Warsaw University of Technology: Warsaw, 2002.
26. ABAQUS online documentation, version 6.5, 2007. Available from: http://129.25.16.135:2080/v6.5/.
27. Guangyao L, Belytschko T. Element-free Galerkin method for contact problems in metal forming analysis. *Engineering Computations* 2001; **18**:62–78.
28. Liu GR, Gu YT. A local radial point interpolation method (LRPIM) for free vibration analyses of 2-D solids. *Journal of Sound and Vibration* 2001; **246**:29–46.
29. Liu GR, Dai KY, Lim KM, Gu YT. A radial point interpolation method for simulation of two-dimensional piezoelectric structures. *Smart Materials and Structures* 2003; **12**:171–180.
30. Hoover WM, Hoover CG, Merritt EC. Smooth-particle applied mechanics: conservation of angular momentum with tensile stability and velocity averaging. *Physical Review E* 2004; **69**(1):016702.
31. Hieber SE, Koumoutsakos P. A Lagrangian particle method for the simulation of linear and nonlinear elastic models of soft tissue. *Journal of Computational Physics* 2008; **227**(21):9195–9215.