Real-Time Nonlinear Finite Element Analysis for Surgical Simulation Using Graphics Processing Units[*]

Zeike A. Taylor[1,2], Mario Cheng[1], and Sébastien Ourselin[1]

[1] BioMedIA Lab, e-Health Research Centre, CSIRO ICT Centre, Level 20, 300 Adelaide St, Brisbane, QLD, 4000, Australia
[2] Centre for Medical Image Computing, University College London, Gower St, London, WC1E 6BT, UK
z.taylor@cs.ucl.ac.uk, {Mario.Cheng, Sebastien.Ourselin}@csiro.au

Abstract. Clinical employment of biomechanical modelling techniques in areas of medical image analysis and surgical simulation is often hindered by conflicting requirements for high fidelity in the modelling approach and high solution speeds. We report the development of techniques for high-speed nonlinear finite element (FE) analysis for surgical simulation. We employ a previously developed nonlinear total Lagrangian explicit FE formulation which offers significant computational advantages for soft tissue simulation. However, the key contribution of the work is the presentation of a fast graphics processing unit (GPU) solution scheme for the FE equations. To the best of our knowledge this represents the first GPU implementation of a nonlinear FE solver. We show that the present explicit FE scheme is well-suited to solution via highly parallel graphics hardware, and that even a midrange GPU allows significant solution speed gains (up to 16.4×) compared with equivalent CPU implementations. For the models tested the scheme allows real-time solution of models with up to 16000 tetrahedral elements. The use of GPUs for such purposes offers a cost-effective high-performance alternative to expensive multi-CPU machines, and may have important applications in medical image analysis and surgical simulation.

1 Introduction

The accurate simulation of tissue deformations arising during surgical procedures presents a formidable modelling challenge [1]. The constitutive behaviour of soft tissues is well known to be nonlinear and time-dependent [2], and the ability of these tissues to undergo large deformations without damage means that linear small strain kinematic formulations are not strictly valid. In many applications, notably surgical simulation [3,4] and intraoperative non-rigid medical image registration [5,6,7], there is a requirement for rigorous modelling of nonlinear deformation. The most physically consistent procedure for estimating such deformations is to use differential equations of continuum mechanics,

[*] This work was performed while the first author was with the BioMedIA Lab.

solved using a numerical technique such as the finite element (FE) method [8]. However a major drawback of such procedures, especially nonlinear ones, is the significant computation times that may be required to solve large models. In many cases successful employment of simulation results depends crucially on the speed with which the results can be obtained. In the case of intraoperative image registration, extended delays while a patient is in the operating position are unacceptable, while interactive surgical simulations require solution at haptic feedback rates (>500Hz).

An efficient total Lagrangian explicit dynamic (TLED) FE algorithm was recently proposed for this purpose [9]. The main advantage of the formulation was that spatial derivatives were referred to the initial configuration of the body under analysis, and could therefore be precomputed. The use of explicit time integration also allowed calculations to be performed in an element-wise fashion, and so avoided the need for solution of large systems of algebraic equations, and allowed for easy incorporation of elaborate constitutive models. As will be seen, it is precisely this feature which renders the algorithm highly suitable for parallel execution.

In recent years there has been a growing body of work concerned with use of graphics processing units (GPUs) for general purpose computations [10]. By appropriately abstracting the graphics computational pipeline and the data upon which it operates, GPUs may be viewed as highly parallel computational engines. As a result GPU implementations of a range of non-graphics algorithms have been presented, for example including linear algebraic applications, ordinary and partial differential equation solvers, image analysis applications, and others (see review by Owens et al. [10]).

In this paper we present an efficient GPU implementation of the nonlinear TLED algorithm, suitable for simulation of soft tissues. The algorithm accounts for all geometric nonlinearities associated with the large deformations which occur in a surgical scenario. Importantly, we achieve significant improvements in solution speed compared with an equivalent CPU implementation, meaning that models of a useful size may be solved in real-time[1]. The significance of the results for purposes of surgical simulation and non-rigid image registration are discussed.

2 Total Lagrangian Explicit Dynamic Framework

Our implementation is based on the TLED algorithm described in [9]. Further details may be obtained from [8] also. The essential steps (from the point of view of GPU implementation) of the algorithm are as follows.

1. Precompute element shape function derivatives $\partial\mathbf{h}$ and mass matrix $\mathbf{M}$.
2. During each time step, n:
   - Apply loads (displacements) and boundary conditions to relevant nodal degrees of freedom.

[1] In which solutions for a time step are obtained in less time the size of the step itself.

– Loop over elements and compute the deformation gradient $\mathbf{X}^n$, strain-displacement matrix $\mathbf{B}_L^n$, 2nd Piola-Kirchhoff stresses $\mathbf{S}^n$, and element nodal forces $\hat{\mathbf{F}}^n$, and add these forces to the total nodal forces $\mathbf{F}^n$.
– Loop over nodes and compute new displacements $\mathbf{U}^{n+1}$ using the central difference method.

A detailed discussion of the formulation and computation of element matrices for various element topologies is presented in [8].

3 GPU Implementation of the TLED Algorithm

As is apparent from Section 2 the TLED algorithm consists of a precomputation phase and a time-loop phase. Since the precomputation phase is performed off-line and only once, our approach was to precompute relevant variables using standard CPU execution, load these variables into textures, and perform all time-loop computations using the GPU. Textures are arrays of values stored in GPU memory which are accessible by the fragment processors during a render pass. Individual texture elements are referred to as *texels*. Each texel may store up to four floating point values, representing red, blue, and green pixel colour values, plus a transparency value (RGBA). By maintaining all simulation variables (displacements, forces, etc) on the GPU itself, we minimise the amount of (time-consuming) CPU-GPU communication that takes place during the simulation.

The time-loop is executed as two render passes (RP1 and RP2), corresponding to the element- and node-loops described in Section 2. Computations are performed on the GPU's *fragment processors*. Loading is applied by prescribing displacements of loaded nodes.

We use a linear tetrahedral element formulation [8]. While these elements are known to produce inferior results when used for simulation of nearly incompressible materials [11], from a data structure point of view they provide significant advantages for GPU implementation. Since each element comprises four nodes, and each node posseses three degrees of freedom, all element matrices have dimensions which are multiples of these values, and are therefore very convenient for storage in four channel (RGBA) texels. Other element topologies such as hexahedra could be implemented in the same way, but all element matrices would be doubled in size and require twice as many texture reads. Additionally, recent efforts have produced 4-node tetrahedron formulations which overcome the locking phenomenon to a large extent [12]. For these reasons we do not feel that use of linear tetrahedra significantly diminishes the contributions of this first GPU development.

Material constitutive response is modelled using a Neo-Hookean model [8].

3.1 Render Pass 1: Element Loop

The purpose of RP1 is to compute element nodal force contributions $\hat{\mathbf{F}}^n$. All precomputed element data are stored in 2D textures on the GPU prior to commencement. The dimensions of the textures are calculated such that the number of texels equals the number of elements. At each time step, prior to execution of RP1, the GPU viewport is reset to these dimensions also to ensure the correct number of pixels are rendered. As the GPU renders the viewport the coordinates for each pixel are generated by the rasterizer and passed to the fragment processors [13]. By maintaining the same scale for rendered pixels and element-associated textures the coordinates of a pixel can be directly used to reference any data relevant to the corresponding element. This is known as 1:1 mapping and is commonly used in general purpose GPU applications.

Computation of element nodal forces requires the element nodal displacements, shape function derivatives, and volume. The latter two may be accessed from textures directly using the generated pixel coordinates. Displacements are accessed via a lookup texture, in which each texel contains the four node indices (from which displacement texture coordinates may be computed) for the current element. With these data retrieved the element nodal forces may be computed using the procedure decribed in [9].

For each element 12 force values are produced at each time step. The force contributions from each element must be summed to obtain the total forces on each node. Ideally, the computed forces would be added to a global nodal force vector as they are computed, but this would involve random texture writes (so-called *scatter operations*) which are currently prohibited. Therefore, force values computed in RP1 are written to textures, which are subsequently read and summed during the second render pass (node loop). The scatter operation is therefore reformulated as a *gather*. Four force textures are attached to the frame buffer to accommodate the four nodal force vectors produced by each element.

3.2 Render Pass 2: Node Loop

In RP2 the element nodal forces computed in RP1 are summed and used to compute new nodal displacements $\mathbf{U}^{n+1}$. The next set of imposed displacements on contacted nodes are also applied. The viewport is reformatted to dimensions equivalent to the number of nodes in the system in order to achieve a 1:1 mapping with the node-associated textures attached during this pass. Additionally, the four force textures rendered during RP1 are reattached as an array of input textures for RP2.

The major task of the fragment program for RP2 is the gathering of the element nodal force contributions for the current node. The locations and numbers of the relevant force values in the four force textures are determined via two lookup textures, labelled `NodeElCrds` and `FCrds`. The latter contains lists of force texture coordinates for each node in the system. Since node valencies are not constant, it is not possible (or is very inefficient) to structure this texture with a fixed mapping to the viewport dimensions. Therefore texture `NodeElCrds` (which does have the dimensions of the viewport), is used to provide the location of the first force texture coordinate in `FCrds` for the current node, and also the valence of this node. The fragment program then fetches and sums the required number of force values from the force textures using a dynamic loop structure. It then remains to compute $\mathbf{U}^{n+1}$, as mentioned.

4 Performance of the Algorithm

In order to assess the performance of the GPU implementation, we analysed two configurations: a cube model undergoing stretching and a brain model subject to indentation (see Fig. 1). The first served to illustrate the relative speed improvement achieved with the GPU, while the second constituted an example of relevance to both interactive surgical simulation and non-rigid image registration. In both cases we compared the solution times per time step for the GPU implementation with that of an equivalent CPU implementation. The GPU version was coded using Cg [14] and OpenGL [13], while the CPU version was written using C++. The test machine included a single 3.2GHz Intel P4 CPU and 2GB of RAM. An NVIDIA GeForce 7900GT GPU (550MHz clock speed, 512MB RAM) was used. Solution times were obtained for a range of mesh densities for each model. In each case five simulations were run, and the mean solution times are reported.

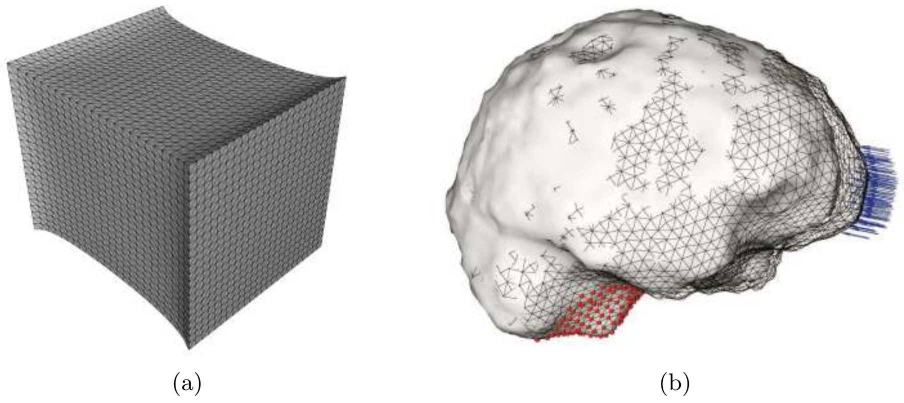
(a) (b)

Fig. 1. (a) Cube model (82944 elements) after stretching by 20%. (b) Overlaid images of the undeformed (wire-frame) and deformed (surface) brain model (46655 elements). Anchor nodes near the brain stem are identified by spheres. Locations of displaced nodes and their displacement directions are indicated by arrows. Computation times for these mesh densities were 4.88ms/time step and 3.46ms/time step, respectively.

4.1 Stretching of a Cube Model

A series of cube models with edge lengths of 0.1m were constructed. Meshes with from 6000 to 82944 elements were used. The models were used to simulate stretching (by 20%) of a cube in which the displaced face and its opposing face were assumed to be fixed to loading platens. Lamé parameters were $\lambda = 49329$Pa and $\mu = 1007$Pa, chosen to correspond to a generally accepted stiffness value for brain tissue of $E = 3000$Pa, with a Poisson ratio of $\nu = 0.49$ (approximating incompressibility) [9]. Mass density was assumed to be that of water, i.e. $\rho = 1000\text{kg/m}^3$. It should be noted that the critical time step size for explicit analyses such as these is dependent on the material parameters used and the minimum characteristic element length in the mesh [8]. For this reason we report the solution times *per time step*, which are independent of these.

The GPU-computed deformed shape is shown in Fig. 1(a). The CPU and GPU solution results were identical in all respects. The mean solution times and ratios of solution times for each model size are plotted in Figs. 2(a) and (b), respectively. The solution times scaled approximately linearly with model size for both CPU and GPU implementations, which may be expected since the main computational effort of the algorithm is the *element-wise* computation of nodal forces $\hat{\mathbf{F}}^n$. A decisive speed improvement (up to approximately 16.4×) was achieved with the GPU implementation, as shown in Fig. 2(b).

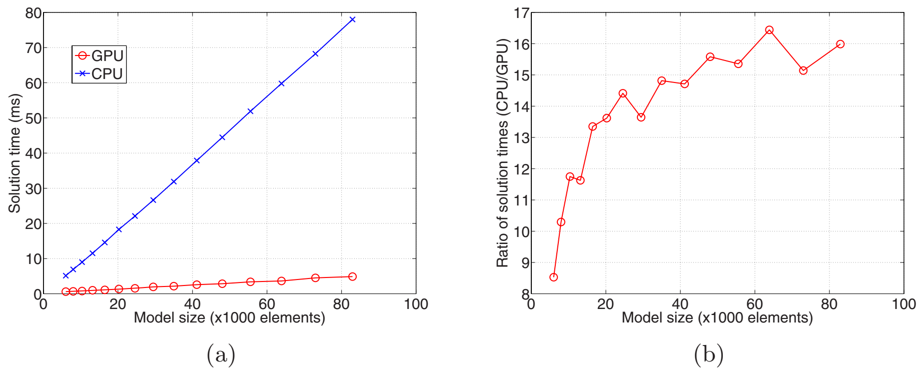

Fig. 2. (a) Mean solution times, and (b) ratios of CPU to GPU solution times for a single time step for stretching of a cube model, plotted against model size

4.2 Indentation of a Brain Model

The next set of tests involved simulation of the indentation of a brain model. The brain geometry was obtained from segmented magnetic resonance images used in the study by Clatz et al. [15]. Three meshes were generated from these data, with 11168, 25017, and 46655 elements, respectively. Material parameters were as used in the cube models. Anchor nodes (with all finite element degrees of freedom fixed) were selected from near the brain stem, while displaced nodes were selected from around the frontal lobes (see Fig. 1(b)). Displacements of 0.01m were applied. This configuration was not intended to represent any particular surgical scenario, but was presented as a generic example of “neurosurgical-type” deformations.

The undeformed (wire-frame) and GPU-computed deformed (surface) shapes are superimposed in Fig. 1(b). Again, CPU and GPU results were identical. The solution times, and ratios of solution times are plotted in Figs. 3(a) and (b), respectively. Again, significant speed improvements were observed (though somewhat less than for equivalent sized cube models), with a peak value of approximately 14×.

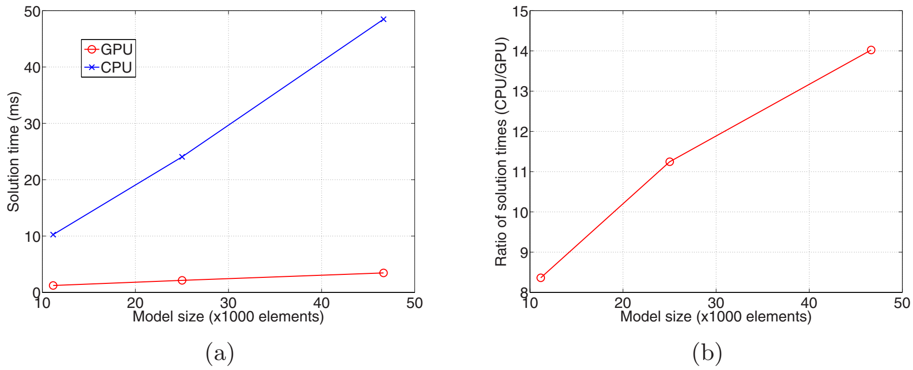

Fig. 3. (a) Mean solution times, and (b) ratios of CPU to GPU solution times for a single time step for indentation of a brain model, plotted against model size

5 Discussion and Conclusions

A novel GPU implementation of an efficient nonlinear FE algorithm suitable for soft tissue deformation simulation has been presented. It was shown that the algorithm employed was well suited to parallel execution, meaning solution speed improvements compared with an equivalent CPU implementation of greater than an order of magnitude were achieved. The largest achieved speed improvement was 16.4×, and we were able to achieve real-time solution of models of up to approximately 16000 tetrahedral elements. To the best of our knowledge this represents the first attempt to port a fully nonlinear FE algorithm to the GPU. Since the algorithm includes both kinematic and constitutive nonlinearities it is suitable for simulation of soft tissue deformation, for example in applications such as interactive surgical simulation and intra-operative non-rigid medical image registration. In view of the substantial speed gains achieved with this first GPU implementation we feel the present development is of great significance to such time-critical applications as these.

Use of biomechanical modelling in surgical simulation and image-guided therapy applications is becoming increasingly widespread. The large body of knowledge and techniques developed by the biomechanics community over many decades provide a powerful basis for addressing many problems in these areas. In many such applications there is a fundamental difficulty in reconciling conflicting requirements for modelling fidelity and expeditious solution. The rapid development of GPU technology has raised the possibility of an entirely new paradigm in high performance computing. The present contribution of a nonlinear FE algorithm implemented for GPU execution shows that this computing model provides a low cost means for performing realistic biomechanical simulations at speeds at or near real-time. We feel that this development has many applications in the areas mentioned.

Acknowledgements

The authors thank P. Raniga for helpful discussions and assistance with GPU programming, and B. Joshi for providing brain meshes. They also thank K. Miller and G. Joldes for much assistance with details of the TLED algorithm.

References

1. Miller, K., Taylor, Z., Nowinski, W.L.: Towards computing brain deformations for diagnosis, prognosis and neurosurgical simulation. Journal of Mechanics in Medicine and Biology 5(1), 105–121 (2005)
2. Taylor, Z.A., Miller, K.: Constitutive modelling of cartilaginous tissues: A review. Journal of Applied Biomechanics 22(3), 212–229 (2006)
3. Szekely, G., Brechbühler, C., Hutter, R., Rhomberg, A., Ironmonger, N., Schmid, P.: Modelling of soft tissue simulation for laparscopic surgery simulation. Medical Image Analysis 4, 57–66 (2000)
4. Picinbono, G., Delingette, H., Ayache, N.: Non-linear anisotropic elasticity for real-time surgery simulation. Graphical Models 65, 305–321 (2003)
5. Carter, T.J., Sermesant, M., Cash, D.M., Barratt, D.C., Tanner, C., Hawkes, D.J.: Application of soft tissue modelling to image-guided surgery. Medical Engineering & Physics 27(10), 893–909 (2005)
6. Wittek, A., Miller, K., Kikinis, R., Warfield, S.K.: Patient-specific model of brain deformation: Application to medical image registration. Journal of Biomechanics 40(4), 919–929 (2007)
7. Tanner, C., Schnabel, J.A., Hill, D.L.G., Hawkes, D.J., Leach, M.O., Hose, D.R.: Factors influencing the accuracy of biomechanical breast models. Medical Physics 33(6), 1758–1769 (2006)
8. Bathe, K.J.: Finite Element Procedures. Prentice Hall, Upper Saddle River, N.J (1996)
9. 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 23(2), 121–134 (2007)
10. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26(1), 80–113 (2007)
11. Hughes, T.J.R.: The Finite Element Method: Linear Static and Dynamic Finite Element Analyses. Prentice-Hall, Inc. Englewood Cliffs, NJ (1987)
12. Bonet, J., Marriott, H., Hassan, O.: An averaged nodal deformation gradient linear tetrahedral element for large strain explicit dynamic applications. Communications in Numerical Methods in Engineering 17(8), 551–561 (2001)
13. Shreiner, D., Woo, M., Neider, J., Davis, T.: OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2, 5th edn. Addison-Wesley, Upper Saddle River, NJ (2006)
14. Fernando, R., Kilgard, M.J.: The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics. Addison-Wesley Professional, Castleton, New York (2003)
15. Clatz, O., Delingette, H., Bardinet, E., Dormont, D., Ayache, N.: Patient-specific biomechanical model of the brain: application to parkinson's disease procedure. In: Ayache, N., Delingette, H. (eds.) IS4TM 2003. LNCS, vol. 2673, pp. 321–331. Springer, Heidelberg (2003)