Efficient nonlinear FEM for soft tissue modelling and its GPU implementation within the open source framework SOFA

Olivier Comas[1,3], Zeike A. Taylor[2], Jérémie Allard[3], Sébastien Ourselin[2], Stéphane Cotin[3] and Josh Passenger[1]

1. BioMedIA Lab, The Australian e-Health Research Centre, Brisbane, Australia
2. University College London, London, United Kingdom
3. INRIA Alcove, Lille, France

Abstract. Accurate biomechanical modelling of soft tissue is a key aspect for achieving realistic surgical simulations. However, because medical simulation is a multi-disciplinary area, researchers do not always have sufficient resources to develop an efficient and physically rigorous model for organ deformation. We address this issue by implementing a CUDA-based nonlinear finite element model into the SOFA open source framework. The proposed model is an anisotropic visco-hyperelastic constitutive formulation implemented on a graphical processor unit (GPU). After presenting results on the model's performance we illustrate the benefits of its integration within the SOFA framework on a simulation of cataract surgery.

1 Introduction

The field of medical simulation is expanding rapidly. Its multi-disciplinary aspect requires the integration within a single environment of solutions in areas as diverse as visualisation, biomechanical modelling, haptic feedback and contact modelling. This diversity of problems creates challenges for researchers to advance specific areas, and leads rather often to duplication of effort. The Open Source SOFA framework [1] was created to overcome this issue by providing researchers with an advanced software architecture that facilitates the development of new algorithms and simulators. One of the main criteria used for assessing the level of realism within a simulator is the ability for the latter to reproduce the deformation of anatomical structures with high fidelity. Therefore, accurate and efficient soft tissue simulation is a critical concern in surgical simulation. For simplicity, most analyses were initially based on linear formulations in order to satisfy real-time constraints[2,3]. More recently, co-rotational models [4] allowing geometric non-linearities have been introduced to overcome some of the limitations of previous formulations. However, the constitutive law in all these models remain linear. Recently, a nonlinear finite element algorithm was introduced in [5]: the Total Lagrangian Explicit Dynamic algorithm (TLED), which was later implemented on Graphics Processing Units (GPU) [6,7]. This implementation allowed real-time simulation of soft-tissue deformation with large meshes and a gain in computational performance of more than 16 times what can be achieved on the CPU.

In this paper we describe our CUDA-based re-implementation of the TLED algorithm within the international and Open Source framework SOFA. We show that this integration has a very limited cost in terms of performance by comparing the SOFA version with a standalone implementation. By providing an efficient and accurate nonlinear FEM for soft tissue modelling to worldwide researchers,we thus hope to assist in enhancing the realism of medical simulators. Reciprocally this integration into the SOFA framework benefits from additional features, which we illustrate through an example: the rapid prototyping of a cataract surgery simulator using the TLED algorithm to simulate the deformation of the lens.

2 SOFA, an open source simulation framework

2.1 Objectives

As previously mentioned, research and development in medical simulation requires many diverse skills. Although their interaction is essential to design a realistic simulator, only few teams have the sufficient resources to build such frameworks. SOFA is an open source framework which aims to answer these challenges. SOFA has been mostly developed by INRIA (the French national institute for research in computer science and control) and CIMIT (Center for Integration of Medicine and Innovative Technology) and is primarily targeted at real-time simulation with an emphasis on medical simulation. SOFA is highly modular and flexible: it allows independently developed algorithms to interact together within a common simulationwhile minimising the development time required for integration [1]. The overall goal is to develop a flexible framework while minimising the impact of this flexibility on the computation overhead. To achieve these objectives, SOFA proposes a new architecture that implements a series of concepts described below.

2.2 SOFA architecture

High-level modularity. The SOFA architecture relies on the innovative notion of multi-model representation where an object is explicitly decomposed into various representations: Behaviour Model, Collision Model, Collision Response Model, Visual Model and Haptic Model. Each representation can then be optimised for a particular task (biomechanics, collision detection, visualisation, haptics) while at the same time improving interoperability by creating a clear separation between the functional aspects of the simulation components. These representations are then connected together via a mechanism called *mapping*. Various mapping functions can be defined, and each mapping associates a set of primitives of a representation to a set of primitives in the other representation (Figure 1). For instance, a mapping can connect degrees of freedom in a Behaviour Model to vertices in a Visual Model.

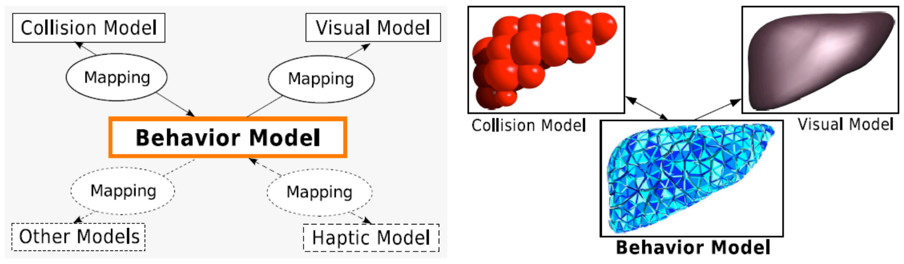


Fig. 1. Multi-model representation in SOFA. Left: a Behaviour Model controls the other representations via a series of mappings. Right: examples of representations with a liver model.

Finer level modularity. In order to easily compare algorithms within SOFA, more flexibility was added to the Behaviour Model by introducing an even finer level of granularity. A series of generic primitives common to most physics-based simulations have been defined: DoF, Mass, Force Field and Solver. The DoF component describes the degrees of freedom, and their derivatives, of the object. The Mass component represents its mass. The Force Field describes both internal and external forces that can be applied to this object. The Solver component handles the time step integration, i.e. advancing the state of the system from time t to time $t + \Delta t$.

Scene-graph. Finally, another key aspect of SOFA is the use of a scene-graph to organise and process the elements of a simulation. Each component is attached to a node of a tree structure. This simple structure makes it easy to visit all or a subset of the components in a scene, and dependencies between components are handled by retrieving sibling components attached to the same node. During the simulation loop, most computations can be expressed as a traversal of the scene-graph. For instance, at each time step, the simulation state is updated by processing all Solver components, which will then forward requests to the appropriate components by recursively sending actions within its sub-tree.

These different functionalities and levels of abstraction allow the user to switch from one component to another by simply editing an XML file, without having to recompile. In particular this permits testing of different computational models of soft tissue deformation, and to assess the pros and cons of various algorithms within the same context.

3 TLED and its GPU implementation into SOFA

3.1 TLED overview

The TLED algorithm is a geometrically and materially nonlinear dynamic finite element method. A nonlinear kinematic framework valid for large deformation is used along with nonlinear constitutive formulations. Both damping and inertial terms are included in the equations of equilibrium, and explicit time integration is employed. A full description of the TLED algorithm can be found in previous publications [5,6,7]. Briefly, the procedure is divided into precomputation and time loop phases. During the simulation loop we proceed as follows:

1. apply imposed displacements and boundary conditions,
2. for each element compute
   (a) displacement derivatives and deformation gradient,
   (b) right Cauchy-Green deformation tensor and 2^{nd} Piola-Kirchhoff stress,
   (c) strain-displacement matrix,
   (d) element nodal force contributions and add these forces to the global nodal forces, and
3. for each node compute new displacements using the central difference method.

3.2 Anisotropic viscoelastic constitutive equation

Viscoelasticity (time dependence) and anisotropy (direction dependence) are well known features of the response of most soft tissues but they are often neglected for computational efficiency. However, because the element stresses are computed directly from strains in explicit analyses, elaborate constitutive models may be incorporated with relative ease. In the present work we used a transversely isotropic visco-hyperelastic model with preferred material direction defined by the unit vector **a**. To the best of our knowledge, this is the first time that such a formulation has been implemented on GPU.

The model is defined in terms of a time-dependent strain energy function

$$\hat{\Psi} = \int_0^t [1 - \alpha(1 - e^{(s-t)/\tau})]\partial_s \Psi^{iso} + \Psi^{vol}, \tag{1}$$

where α and τ are viscoelastic parameters, t is time, and ∂_s denotes partial differentiation with respect to the dummy variable s. Ψ^{iso} and Ψ^{vol} are the isochoric and volumetric components, respectively, of the underlying hyper-elastic strain energy function:

$$\Psi^{iso} = \frac{\mu}{2}(\bar{I}_1 - 3) + \frac{\eta}{2}(\bar{I}_4 - 1)^2, \qquad \Psi^{vol} = \frac{\kappa}{2}(J-1)^2, \tag{2}$$

where μ, η and κ are material parameters J is the determinant of the deformation gradient $\mathbf{F}$, $\bar{I}_1$ is the first invariant of the modified right Cauchy-Green tensor $\bar{\mathbf{C}} = J^{-2/3}\mathbf{F}^T\mathbf{F}$, and $\bar{I}_4 = \mathbf{a} \cdot \bar{\mathbf{C}}\mathbf{a}$ is a pseudo-invariant of $\bar{C}$ and $\mathbf{a}$. We consider only viscoelastic isochoric terms.

The 2^{nd} Piola-Kirchhoff stress $\mathbf{S}$ is obtained through differentiation of (1) with respect to strain. Defining $\Upsilon = \int_0^t 1 - \alpha(1 - e^{(s-t)/\tau})2\partial_{\mathbf{C}_s}\Psi^{iso} ds$ we may obtain $S = 2\partial_{\mathbf{C}}\hat{\Psi} = 2\partial_{\mathbf{C}}\Psi^{iso} + 2\partial_{\mathbf{C}}\Psi^{vol} - \Upsilon$. We may then update stresses at each step n using

$$\mathbf{S}_n = 2(\partial_{\mathbf{C}} \Psi^{iso})_n + 2(\partial_{\mathbf{C}} \Psi^{vol})_n - \Upsilon_n \tag{3}$$

where $(\partial_{\mathbf{C}} \Psi^{iso})_n$ and $(\partial_{\mathbf{C}} \Psi^{vol})_n$ may be computed from the known current deformation $\mathbf{C}_n$. Finally it may be shown that Υ_n is computable from

$$\Upsilon_n = \frac{2\Delta t \alpha (\partial_{\mathbf{C}} \Psi^{iso})_n + \tau \Upsilon_{n-1}}{(\Delta t + \tau)}. \tag{4}$$

3.3 CUDA description

GPUs achieve a high floating point capacity by distributing computation across a high number of parallel execution threads. They perform optimally as Single Instruction, Multiple Data devices. CUDA is a relatively new C API for compatible NVIDIA GPUs. CUDA organises threads in two hierarchical levels: blocks, which are groups of threads executed on one of the GPU's multiprocessors, and grids, which are groups of blocks launched concurrently on the device, and which all execute the same kernel. Figure 2 represents this thread organisation. As an example, the NVIDIA 8800 GTX used for the results presented in section 4 has 16 multiprocessors, each containing eight processors.

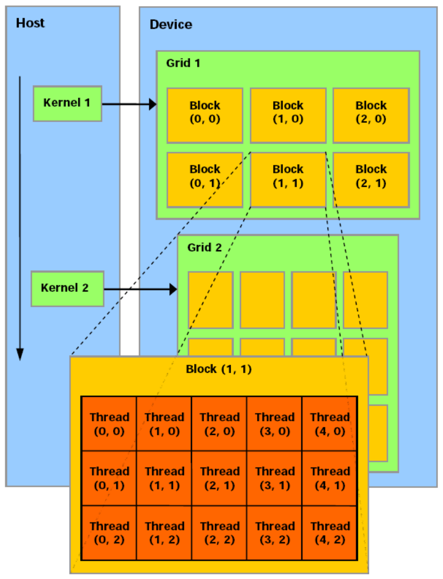


Fig. 2. Each kernel is executed by CUDA as a group of threads within a grid. Image courtesy of NVIDIA [8].

CUDA allows developers to specify the number of threads per block in each execution (the so-called execution configuration), effectively defining the distribution of computational load across all processors. For a given kernel the block dimensions are chosen to optimise the utilisation of the available computational resources. Care should be taken at the multiprocessor level in balancing the available memory required by the kernels with the ability to hide global memory latency. Since a finite amount of memory is available on a multiprocessor, the memory requirements of a kernel will determine how many threads can run concurrently on each. Importantly, CUDA's use of time slicing allows more than one block to be executed concurrently on a single multiprocessor, which has important implications for hiding memory latency. If more than one block is executing, the multiprocessor is able to switch processing between blocks while others are stalled on memory accesses, whereas it has no option but to wait for these if only one block is executing. Therefore for memory bandwidth bound kernels it may be preferable to launch several smaller blocks on each multiprocessor rather than a single larger one if both configurations make the same use of multiprocessor memory resources. While tools are available from NVIDIA for estimating the optimal execution configuration, it has proved necessary to fine tune the configuration experimentally for each kernel.

3.4 GPU implementation into SOFA

SOFA integration. Implementing a biomechanical model in SOFA translates essentially into writing a new Force Field, i.e. describing the algorithm used to compute internal forces in the model. It merely comes down to creating a single C++ class and changing the position reads and force writes to integrate the algorithm into SOFA's design. The precomputation phase takes place in the initialisation method where relevant variables are computed and passed to an external C function that allocates memory on the GPU and binds textures to it. During the simulation loop, the Solver requests the computation of the forces by launching the appropriate kernels on the GPU.

Kernel organisation. The TLED GPU implementation relies on 2 kernels. The first kernel operates over elements in the model and computes the element stresses based on the current model configuration. It then converts these into nodal force contributions, which are written to global memory. The second kernel operates over nodes and reads the previously calculated element force contributions and sums them for each node. The SOFA central difference solver computes the displacements from the nodal forces. The element force contributions could directly be added to a global force on each node at the end of the first kernel, but this would involve scattered writes. Although CUDA allows scattered writes, it offers no write conflict management between threads, and potential measures to address this severely affect the performance. Therefore, due to the impracticability of scattered writes, the sum operation is reformulated as a gather and the second kernel is needed to sum the nodal forces.

Memory usage. One efficient method for reading global memory data within kernels is texture fetching. Textures may be bound to either *cudaArrays* or regions of linear memory. CudaArrays have been designed to achieve optimal fetching when the access pattern has a high level of 2D locality. In the present application, the access pattern among threads is essentially random (since unstructured meshes are used) and our experiments have shown that texture fetching from linear memory is in fact fastest. Therefore all global memory variables were accessed using this method.

In SOFA, the forces are stored on the GPU in global memory. Since this memory space is not cached, it is important to follow the appropriate access pattern to obtain maximum memory bandwidth, especially given how costly accesses to device memory are. A multiprocessor takes 4 clock cycles to issue one memory instruction for a set of threads. When accessing global memory, there are, in addition, 400 to 600 clock cycles of memory latency. A suboptimal access pattern would yield incoherent writes. The memory bandwidth would then be an order of magnitude lower. In order to prevent this, a key feature of CUDA has been used: *shared memory*. This is a very fast memory shared by all the processors of a multiprocessor. The results of the second kernel are first copied to shared memory and then moved to global memory. If the copies are well organised, it is possible to re-order the access to fulfil all the memory requirements (for both shared and global) and thus reach the maximum bandwidth.

CPU-GPU interaction. CPU-GPU interaction is generally a significant bottleneck in General Purpose GPU applications due to the relatively low interface bandwidth and it is desirable to minimise such interaction. However, interaction cannot be entirely removed from the present implementation since, for example, the solver requires inputs in the form of loaded nodes (which may change due to the interaction with the user) and their displacements, and may need to provide outputs in the form of reaction forces for haptic feedback. CUDA alleviates the problem somewhat by allowing allocation of areas of page-locked host memory which are directly accessible by the GPU and therefore offer much higher bandwidth. In SOFA, all transfers between CPU and GPU are made via this mechanism.

Element technology. Tetrahedral meshes are easily generated and therefore widely used in simulations. Although we used 4-node linear tetrahedra in our previous implementation [6,7], these are known to be susceptible to volumetric locking [9]. To overcome this limitation, we added support for reduced integration 8-node hexahedral elements which are preferable both in terms of accuracy and computational efficiency. A drawback of using hexahedra is the existence of so-called Hourglass modes that have to be addressed to avoid deterioration of the solution [10]. Techniques for suppressing these modes exist, but naturally involve additional computations. However, for a given number of degrees of freedom (DOF), a hexahedral mesh can be built with far fewer elements than a tetrahedral one. Since the majority of the calculations in explicit dynamic analyses are performed per element, this results in reduced overall computation time.

From a GPU perspective, hexahedral element computations are substantially heavier and demand more memory resources. Most of the matrices are twice as large for hexahedra, which necessitates twice as many texture fetches per element, and use of twice as many registers per thread. Thus the occupancy (GPU percentage usage) drops from 25% to only 8%. Similarly, twice as many nodal forces per element are written to global memory by the first kernel. Additional variables for hourglass control are also required. Therefore, on a per element basis hexahedra are significantly less efficient than tetrahedra, especially for GPU execution where memory efficiency is crucial. However, from the point of view of an entire model the lower number of hexahedra required for a given number of degrees of freedom still outweighs this element-wise inefficiency.

4 Results

4.1 Performance of the CUDA-based implementation

The algorithm must be efficient to be useful in a real-time environment. Thus we assessed the computational performance of the new CUDA implementation by comparing times to a C++ implementation as we did with our OpenGL-based implementation [6,7]. We generated meshes with between 3 993 and 177 957 DOF and measured the solution times for a single time step on an Intel Core 2 Duo 2.4GHz CPU, 2GB RAM and an NVIDIA GeForce 8800 GTX. Figure 3 shows the substantial speed increase brought by the GPU implementation. We note that the speed improvement factor grows to 53.6×, which makes the CUDA-based implementation approximately 3 times faster than the OpenGL one on current hardware.

4.2 Performance within SOFA

The efficiency and the side-effects of porting the algorithm into the flexible SOFA framework need to be measured. Therefore the performance has been assessed by comparing the computational time of the algorithm running within and outside SOFA. NVIDIA provides a tool to check the GPU implementation by evaluating many variables during the execution like for instance timings, counts of inconsistent reads and writes or GPU occupancy. We used this tool to carry out two measures:

1. GPU time only estimates the GPU computational time.
2. CPU time allows the evaluation of the execution time with the additional overhead due to the framework.

The tests were performed on a simple test scene featuring a cube under gravity. Hexahedral meshes with different resolutions from 1 331 to 29 791 nodes were used and the results are presented in figure 4. In our standalone implementation, the second kernel not only accumulates nodal forces but also adds gravity

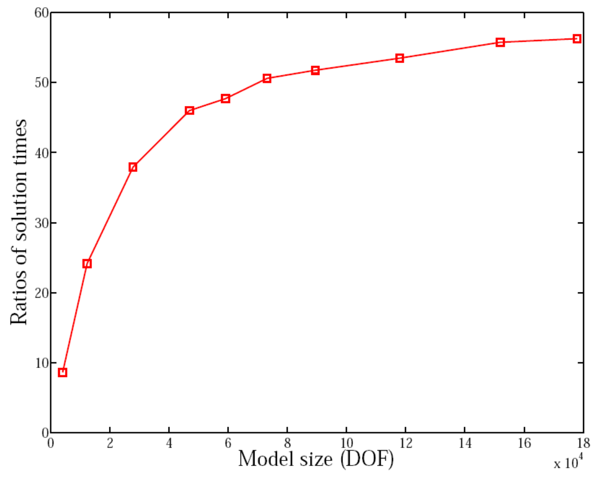


Fig. 3. Ratio of CPU to GPU solution times for the anisotropic viscoelastic formulation.

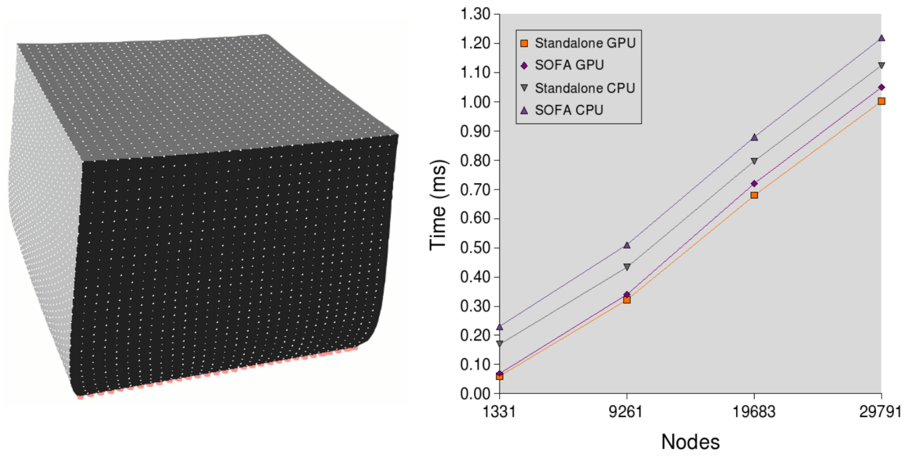


Fig. 4. Left: FEM mesh of cube with 29 791 nodes deformed under a uniformed load. Right: Comparison of GPU computational timings and CPU overheads between the SOFA and standalone implementations for different mesh sizes.

and updates positions based on the central difference integration scheme. These operations are split into separated components in SOFA, in order to introduce more flexibility (such as applying additional forces or changing the integration algorithm). While this introduces no noticeable difference on a CPU-based simulation, when using the GPU it is more costly due to overheads in the CUDA API for the additional kernel launches. Although it reduces the performance by 8.4% for large meshes, this could be optimised away by adding a kernel specific to a given combination of components.

4.3 A medical application: cataract surgery

To illustrate the benefit of integrating this new soft tissue deformation model within the SOFA framework, we created a simulation of phacoemulsification, used for cataract surgery (figure 5). It consists in removing the natural lens and inserting an artificial intraocular lens implant. To remove the natural lens through a very small opening on the side of the cornea, it is emulsified using an ultrasound device and then aspirated from the eye. During the procedure the lens also undergoes many large deformations which need to simulated. The phacoemulsification step itself is represented by removing elements of the volumetric lens mesh. To be realistic, the simulation requires the lens to be finely meshed, which directly impacts the computation of the deformation of the lens. Limited by the processing resources of the CPU, the previous version of the lens model used a rather coarse mesh to achieve interactive simulation rates. Although studies of biomechanical properties of cornea have shown that the mechanical response was nonlinear [11], viscoelastic [12,13] and anisotropic [11,14], the simulation used a linear model. Therefore the physical model underlying the cataract surgery simulation needed to be improved.

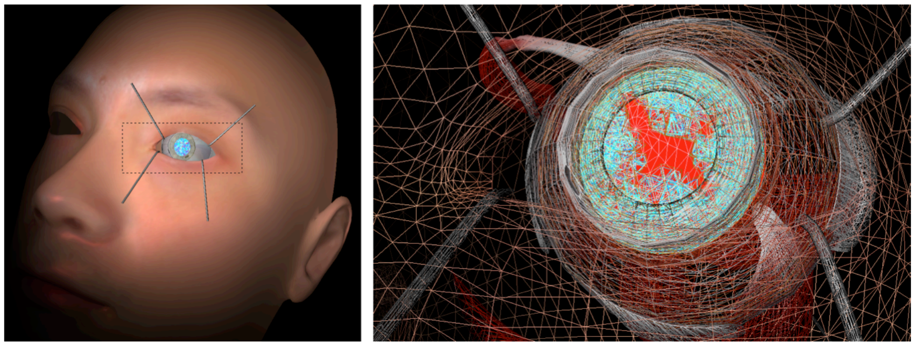
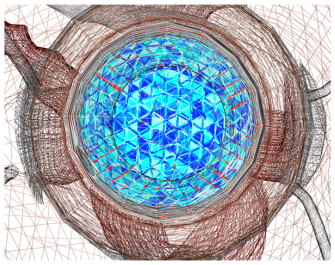


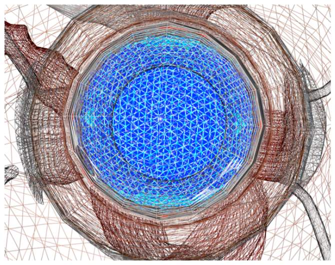
Fig. 5. Cataract surgery simulation. Left: global view of the operating scene. Right: illustration of the complexity of the meshes involved in the simulation.

The flexibility of SOFA allows users to easily change the biomechanical models that are used in a simulation by editing the xml file describing the scene. The

TLED performance has been compared to a co-rotational [4] FEM implementation on the CPU (see Figure 6). The TLED GPU implementation is clearly faster than the CPU version. However, by using an implicit solver the latter is more robust to collision forces. On the other hand, the TLED adds nonlinearity, viscoelasticity and anisotropy to the mechanical response modelling. This application clearly illustrates the benefits of SOFA. By providing a framework where algorithms can easily be exchanged, one can experiment and ascertain the most suitable combination of algorithms for a particular application.



Mesh	FEM CPU	TLED GPU
(a)	37 fps	284 fps
(b)	2.3 fps	87 fps



(a) 814 nodes, 4058 elements

(b) 7826 nodes, 47633 elements

Fig. 6. Comparison of cataract surgery performance using co-rotational FEM on CPU and TLED on GPU.

5 Conclusion

We have implemented an efficient fully nonlinear FEM using CUDA in the open source framework SOFA. Adding a physical solver to SOFA able to model the nonlinear, viscoelastic and anisotropic features of the mechanical response of a material should enhance the fidelity of tissue deformations. We have demonstrated the efficiency of the GPU TLED implementation, and we applied it to cataract surgery simulation. Furthermore we have shown how one can take advantage of SOFA by experimenting with the interaction between algorithms. We hope that our contribution to SOFA will encourage others to share implementations.

References

1. Allard, J., Cotin, S., Faure, F., Bensoussan, P.J., Poyer, F., Duriez, C., Delingette, H., Grisoni, L.: Sofa - an open source framework for medical simulation. In: Medicine Meets Virtual Reality. (2007) 13–18
2. Cotin, S., Delingette, H., Ayache, N.: Real-time elastic deformations of soft tissues for surgery simulation. IEEE Transactions On Visualization and Computer Graphics **5 (1)** (1999) 62–73

3. Wu, W., Heng, P.: An improved scheme of an interactive finite element model for 3D soft-tissue cutting and deformation. Computer Animation and Virtual Worlds **21 (8-10)** (2005) 707–716
4. Felippa, C.A.: A systematic approach to the element independent corotational dynamics of finite elements. Technical Report CU-CAS-00-03, Center for Aerospace Structures (2000)
5. 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) (2007) 121–134
6. Taylor, Z., Cheng, M., Ourselin, S.: Real-time nonlinear finite element analysis for surgical simulation using graphics processing units. (2007) 701–708
7. Taylor, Z., Cheng, M., Ourselin, S.: High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. IEEE Transactions on Medical Imaging **27**(5) (2008) 650–663
8. : NVIDIA Programming Guide 1.1. http://developer.nvidia.com/object/cuda.html
9. Hughes, T.: The Finite Element Method: Linear Static and Dynamic Finite Element Analyses. Prentice-Hall, Inc, Englewood Cliffs, NJ. (1987)
10. Flanagan, D.P., Belytschko, T.: A uniform strain hexahedron and quadrilateral with orthogonal hourglass control. International Journal for Numerical Methods in Engineering **17** (1981) 679–706
11. Buzard, K., Hoeltzel, D.: Biomechanics of the cornea. In: Proceedings of SPIE. Volume 1423. (1991) 70–81
12. Zeng, Y., Yang, J., Huang, K., Lee, Z., Lee, X.: A comparison of biomechanical properties between human and porcine cornea. Journal of biomechanics (2001)
13. Kobayashi, A., Staberg, L., Schlegel, W.: Viscoelastic properties of human cornea. Journal of experimental mechanics (2006)
14. Elsheikh, A., Brown, M., Alhasso, D., Rama, P., Campanelli, M., Garway-Heath, D.: Experimental assessment of corneal anisotropy. Journal of refractive surgery (2008)