Implicit Tensor-Mass solver on the GPU

X. Faure[1,†] and F. Zara[1] and F. Jaillet[1,2] and J-M. Moreau[1]

[1]Université de Lyon, CNRS, Université Lyon 1, LIRIS, SAARA team, UMR5205, F-69622, France
[2]Université de Lyon, IUT Lyon 1, Computer Science Department, F-01000, France

Abstract

The realist and interactive simulation of deformable objects has become a challenge in Computer Graphics. For this, the Tensor-Mass model enables local solving of mechanical equations, making it easier to handle local control, like collisions, tool interaction, etc. In this paper, we propose the GPU implementation of this model to achieve interactive time. Moreover, the use of an implicit integration scheme will help to guarantee stability at any time step, leading to a true alternative to Mass-Spring or Finite Element methods.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation and Virtual Reality I.6.8 [Simulation And Modeling]: Types of Simulation—Parallel

1. Introduction

Following the increasing demand of realism in Computer Graphics, physically-based simulation has become a very active research field over the last decade. This is particularly apparent in cloth and hair animation, medical simulation, interactive entertainment, and more generally in all Virtual Reality applications where animation, interaction or alteration of deformable objets is required in interactive time.

The Tensor-Mass approach, introduced in 1999, is a good candidate to handle deformable objects within the context of interactive complex physically-based simulations. Thus, in the proposed work, we present a GPU implementation of the deformation simulation based on this model using an implicit integration scheme to ensure the unconditional stability of our simulation. The paper is organized as follows: section 2 gives some pointers to previous work about the Tensor-Mass approach and GPU programming of physically-based animation. Section 3 illustrates the main steps of the Tensor-Mass formulation for the computation of forces and the induced movement on the object. Section 4 presents our parallel implementation on the GPU, with a highlight on the implicit integration scheme. Finally, section 5 shows our results and a conclusion may be found in section 6.

† This work is partly financed by the PRRH (Rhône-Alpes Research Program on Hadrontherapy) for ETOILE (National French Hadrontherapy Centre).

2. Previous work

Several methods have been published to model the physical behavior of deformable objects [NMK*06].

Among them, the Finite Element Method (FEM) allows the resolution of the differential equations systems driving the movements of objects [NP05]. It is based on the discretization of each object into elements (tetrahedra, for example), and the global solution is then obtained by solving equations on each element and assembling the results in a global matrix. As the physical properties are directly integrated in this mechanical formulation, realistic simulations of the deformation are possible. But this is generally achieved at the expense of a high computation time, especially when non-linear behaviors are required.

On the other hand, the Mass-Spring System is considered as the most direct and intuitive model. Interactive computation time is easily achieved even for complex scenes, but unfortunately, mass repartition and stiffness parameters of the springs are difficult to adjust.

The Tensor-Mass model was introduced by Delingette, Cotin and Picinbono [CDA00, PDA00, Pic03], and extended by Schwartz [SDR*05]. This model may be considered as an alternative to the FEM method. It combines all the advantages of the previously cited models, that is, the deformation forces are derived from the FEM mechanical formulation, but are next computed locally and iteratively for each discretized element. Besides, this allows to handle more directly and easily topological changes and external interactions as in the case of Mass-Spring Systems.

Hence, several formulations have been proposed to account for various mechanical behaviors: the linear model, based on Hooke's law [CDA00]; the non-linear geometrical model based on the Saint Venant-Kirchhoff's elasticity model [PDA00], or anisotropic material [Pic03]. Moreover, Schwartz [SDR*05] presented another extension for non-linear visco-elastic deformations. In this latter work, some tensors are pre-computed to accelerate the process.

Concerning parallelized physically-based animation on the GPU, work was proposed for non-linear FEM soft tissue modeling based on CUDA [CTAO08], FEM cloth simulation [RNSS*06], implicit FEM solver for deformation simulation [ACF11], or Mass-Springs Systems [GW05]. But, as far as we know, the problem of the parallel implementation has not yet been addressed for the Tensor-Mass model. Thus, in this paper, we propose a GPU implementation of this model considering both linear and non-linear mechanical behaviors (based on Hookean and Saint Venant-Kirchhoff's elasticity models, that are the most employed in interactive simulations).

3. Simulation of the deformation of an object

Continuum mechanics concerns the description of an object moving or deforming under the action of stress. The deformation Φ may be formulated according to the displacement $U(X)$ of a point X of the object by $\Phi(X) = X + U(X)$. The strain-tensor ε enables the quantification of this deformation and depends on the gradient of the deformation defined by $\nabla\Phi = I + \nabla U$. Thus, ε may be expressed according to ∇U and depends on the material's properties. Moreover, the deformation of the object, induced by external forces, generates a deformation energy W which naturally depends on the material. Then, the derivative of this energy gives the corresponding deformation force.

Finally, the movement of the deformable object may be expressed by the following differential equations system:

$$M\,\frac{d^2U}{dt^2} + D\frac{dU}{dt} + KU = F, \tag{1}$$

with U the displacement of the object, and M, D, K the mass, damping and stiffness matrices of the simulated object, respectively.

3.1. Tensor-Mass approach

As explained in the previous section, the Tensor-Mass formulation (TM) represents an alternative to the FEM to solve mechanical equations [CDA00]. As for the FEM, the domain is discretized into several elements. However, the equations are next solved locally, involving the following main steps for each element:

- Discretization of the displacement U_E with the definition of interpolation functions Λ according to the chosen type of elements (hexahedron, tetrahedron, etc.);
- Computation of the deformation energy W_E according to the displacement of the element's nodes;
- Computation of the elasticity force F_E by calculating the derivative of the deformation energy W_E.

Then, dynamical equations based on Newton's laws of motion are formulated to compute the deformation and displacement of the object *via* an integration scheme according to its acceleration.

3.2. Interpolation functions for a P_1 element

The displacement of a point X of coordinates (x, y, z) inside a element of the discretized domain is defined by:

$$U_E(X) = \sum_{j=0}^{n-1} \Lambda_j(X)\, U_j, \tag{2}$$

with n the number of nodes of the element, $\Lambda_j(X)$ interpolation functions, and $U_j = U_{P_j}$ the displacement from its initial position of node P_j, defined by its coordinates (P_{jx}, P_{jy}, P_{jz}). Note that the interpolation functions Λ_j are defined according to the kind of element used for the discretization. Moreover, considering an element of volume Vol_0, the following relationship may be expressed:

$$\sum_{j=0}^{n-1} \Lambda_j(X) = Vol_0. \tag{3}$$

P_1 tetrahedral elements correspond to a subdivision of the domain by setting a node at each vertex ($n = 4$). The interpolation functions are then linear and correspond to the barycentric coordinates of X inside the tetrahedron, for $j = 0..3$:

$$\begin{aligned}\Lambda_j(X) &= \alpha_j \cdot X + \beta_j && (4)\\ &= \alpha_{jx}\, x + \alpha_{jy}\, y + \alpha_{jz}\, z + \beta_j && (5)\end{aligned}$$

with

$$\begin{cases} \alpha_j = (-1)^j (P_{j+2} - P_{j+1}) \times (P_{j+3} - P_{j+1}) \\ \beta_j = (-1)^j P_{j+1} \cdot (P_{j+2} \times P_{j+3}) \end{cases}$$

where P_j corresponds to vector OP_j, with O the origin.

P_1 elements are usually used to simulate an object with a linear relation between the stress σ applied on the object and the induced strain ε. Furthermore, the relationship between the strain ε and the gradient of displacement ∇U may be non-linear. Thus, this kind of element is valid for both Hookean and Saint Venant-Kirchhoff's materials.

3.3. Deformation energy for a P_1 element

Different elasticity models exist depending on the expected mechanical behavior of the material. These models are defined by the formulation of the strain-tensor ε, the energy induced by the deformation and the associated elasticity forces.

Linear mechanical behavior. If the elastic deformation is reversible, *i.e.* the deformable object returns into its initial shape when the stress is removed, the elasticity is considered as *linear*. This assumption is especially valid for small displacements – usually less than 10% of the size of the object. For modeling such behavior, only the linear part of the Green-St Venant strain-tensor has to be considered:

$$\varepsilon_l(X) = \frac{1}{2}\left(\nabla U^T(X) + \nabla U(X)\right). \tag{6}$$

The energy of deformation is then defined by:

$$W_l(X) = \frac{\lambda}{2}\left(\operatorname{tr}\varepsilon_l(X)\right)^2 + \mu \operatorname{tr}\varepsilon_l(X)^2, \tag{7}$$

with λ and μ the Lamé coefficients characterizing the material stiffness, defined by:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)},\ \mu = \frac{E}{2(1+\nu)}$$

with E Young's modulus and ν the Poisson ratio.

Non-linear mechanical behavior. For displacements larger than 10% of the object's size, we have to consider a *non-linear* elasticity behavior. Hyper-elasticity provides a means of modeling such materials. The simplest hyperelastic model is an extension of the linear elastic material, based on Green-St Venant strain-tensor ε_{nl}, and its associated energy W_{nl} defined by:

$$\left\{\begin{array}{lcl} \varepsilon_{nl}(X) & = & \frac{1}{2}\left(\nabla U^T(X) + \nabla U(X) + \nabla U^T(X)\,\nabla U(X)\right) \\ W_{nl}(X) & = & \frac{\lambda}{2}\left(\operatorname{tr}\varepsilon_{nl}(X)\right)^2 + \mu \operatorname{tr}\varepsilon_{nl}(X)^2 \end{array}\right. \tag{8}$$

Generalization. The geometrical characteristics of any element of the discretized domain may be linked to its deformation energy W_E, by:

$$W_E(X) = W_{law}(X)\,\frac{1}{Vol_0}, \tag{9}$$

with Vol_0 its volume and $W_{law}(X)$ defined according to the chosen elasticity model (equation (7) or (8)). Moreover, we may note that the fact to have constant ∇U and ∇U^T for a given P_1 element, would imply that W_E, its deformation energy, is also constant inside the tetrahedron (*i.e.* does not depend on X).

3.4. Forces and derivatives forces computation

Considering an element of the discretized domain, the force applied on its node P_i is given by:

$$F_E(P_i) = \frac{\partial W_E(P_i)}{\partial U_i} \tag{10}$$

for $i \in [0, n-1]$, with $W_E(P_i)$ the energy of deformation of the considered element evaluated at node P_i. But, if the knowledge of the forces is sufficient for an explicit integration scheme, the computation of its derivatives is mandatory when implicit integration is used (see below, equation (15)). So, the derivative of the force is given by:

$$dF_E^j(P_i) = \frac{\partial F_E(P_i)}{\partial U_j}\ ,\ j \in [0, n-1]. \tag{11}$$

Finally, if we consider the whole object and E_{P_i}, the set of elements containing node P_i, the force applied on P_i is defined by:

$$F(P_i) = F_i = \sum_{k \mid E_k \in E_{P_i}} F_{E_k}(P_i) = \sum_{k \mid E_k \in E_{P_i}} \frac{\partial W_{E_k}(P_i)}{\partial U_i}, \tag{12}$$

and its derivative follows, for $j \in [0, n-1]$:

$$dF^j(P_i) = dF_i^j = \sum_{k \mid E_k \in E_{P_i}} dF_{E_k}^j(P_i) = \sum_{k \mid E_k \in E_{P_i}} \frac{\partial F_{E_k}(P_i)}{\partial U_j}. \tag{13}$$

Moreover, an incompressibility constraint may be introduced to guaranty the volume preservation of the object, as suggested by Picinbono for Saint Venant-Kirchhoff's elasticity model [Pic03].

3.5. Time integration methods

Once the forces applied on each node have been computed, Newton's equation governing the movement of the object may be used. At time t, we have:

$$m_i \frac{d^2}{dt^2}U_i(t) = F_i(t) - \kappa \frac{d}{dt}U_i(t) \tag{14}$$

with m_i and F_i the mass and force at node P_i, and κ the environment damping. This equation enables the computation of the acceleration of the object according to the applied forces. This equation is linked to the differential equation system (equation (1)) by considering the matrices M, D and K as diagonal by applying mass and damping on each node of the discretized domain.

A numerical integration scheme is then used to obtain the velocity (according to the acceleration) and position (according to the velocity) of the nodes through the simulation time. The simplest integration method is *Euler's explicit scheme*, with:

$$\left\{\begin{array}{lcl} \frac{d}{dt}U_i(t+h) & = & \frac{d}{dt}U_i(t) + h\frac{d^2}{dt^2}U_i(t) \\ U_i(t+h) & = & U_i(t) + h\frac{d}{dt}U_i(t) \end{array}\right.$$

with h the time step. But to obtain a stable simulation, h has to be reduced, especially when stiffness increases. So, to enable larger time steps, an implicit integration scheme is mandatory. By example, *Euler's implicit scheme* may be defined as:

$$\left\{\begin{array}{lcl} \frac{d}{dt}U_i(t+h) & = & \frac{d}{dt}U_i(t) + h\frac{d^2}{dt^2}U_i(t+h) \\ U_i(t+h) & = & U_i(t) + h\frac{d}{dt}U_i(t+h) \end{array}\right.$$

Noting M, F, V and U the mass matrix, forces, velocities and displacements vectors, respectively, this scheme may be reformulated [BW98]:

$$\underbrace{\left(M - h\frac{\partial F}{\partial V} - h^2\frac{\partial F}{\partial U}\right)}_{A}\underbrace{\Delta V}_{x} = \underbrace{h\,F(t) + h^2\frac{\partial F}{\partial U}V(t)}_{b} \quad (15)$$

with $\Delta V = \frac{d}{dt}U(t+h) - \frac{d}{dt}U(t)$ and $\frac{\partial F}{\partial U}$, $\frac{\partial F}{\partial V}$ the matrices encoding the variation of forces resulting from displacements and velocity change. Consequently, we have to resolve this linear system to obtain ΔV and then update the velocity with $\frac{d}{dt}U(t+h) = \Delta V + \frac{d}{dt}U(t)$ and the displacement with $U(t+h) = U(t) + h\frac{d}{dt}U(t+h)$.

The Conjugate Gradient method, presented in Algorithm 1, is usually favored to solve this linear system with a few number of iterations [BW98]. Note that the sparse matrix $\frac{\partial F}{\partial U}$ has not to be explicitly computed (direct computing of its multiplication by a vector) and that the matrix $\frac{\partial F}{\partial V}$ is null (as only internal forces are considered).

Algorithm 1 Conjugate Gradient Algorithm to solve the $Ax = b$ system from Euler's implicit integration scheme

```
1:  $b = h\,F(t) + h^2\frac{\partial F}{\partial U}V(t)$
2:  $x = 0$
3:  $d = r = b$
4:  $\rho_0 = dot(r,r)$
5:  for i = 1 to $n$ do
6:      $df = \frac{\partial F}{\partial U}d$
7:      $A = Md - h^2 df$
8:      $\alpha = \frac{\rho_{i-1}}{dot(d,A)}$
9:      $x = x + \alpha\, d$
10:     $r = r - \alpha\, A$
11:     $\rho_i = dot(r,r)$
12:     $\beta = \frac{\rho_i}{\rho_{i-1}}$
13:     $d = r + \beta\, d$
14:     if $(\rho_i > \varepsilon^2\, \rho_0)$
15:         break
16: end for
```

4. Implementation in SOFA

The Open Source Framework SOFA was used for the implementation of the Tensor-Mass model. This framework, written in C++ and using the XML script language, enables comparisons between models and methods proposed by research groups in medical simulation (http://www.sofa-framework.org, [ACF*07]).

The objects simulated in SOFA are defined by their topology, geometry and visual model (by loading OBJ files, for example), their mechanical state (position, velocity, acceleration, force stored into vectors), their mechanical behavior (constitutive laws) as well as the collision model. Then, the simulation loop is executed, involving in our case four main steps for each node of the discretized domain: (1) computation of the forces, (2) computation of the acceleration according to Newton's second law of motion, (3) integration of the acceleration to obtain velocity, (4) integration of the velocity to obtain position. Note that Euler's implicit integration scheme is used to unconditionaly guarantee the stability of the simulation.

To integrate the Tensor-Mass model in SOFA, we only need to implement two functions, namely `addForce()` and `addDforce()`, enabling the forces computation and their derivatives (only required for Euler's implicit scheme) for each node. But, as the initial GPU implementation of SOFA is based on CUDA [CTAO08], we also parallelized the different steps of the simulation loop to obtain a GPU implementation using the OpenCL language [SGS10].

4.1. Force computation on the GPU

To parallelize the computation of the forces on the GPU, we divide it into two tasks, involving a set of kernels:

1. First, we compute and store the forces applied on each node of a considered element. This computation (`kernel1`) is made for each element of the domain.
2. Next, we sum these partial forces to obtain the total forces applied on each node, involved by the different elements of the domain. This computation (`kernel2`) is made for each node of the domain.

Moreover, three specific data structures are defined for a discretization of a domain into N elements involving m nodes (with n the number of nodes of each element and N_n the maximal number of neighbor elements for a node):

- `index` (of size $N \times n$) stores the relationship between local and global indexation for each node of each element. For example, `index[e][v]` gives the global indexation for node `v` of element `e`.
- `PartialForce` (of size $N_n \times m \times 3$) enables the storage of 3D coordinates of partial forces for each node considering its global indexation. Thus, `PartialForce[][v]` gives partial forces of node `v` of the domain.
- `TotalForce` (of size $m \times 3$) stores the sum of the partial forces for each node considering its global indexation. Thus, `TotalForce[v]` gives the forces of node `v` of the domain.
- `ForceIndex` (of size $N \times n$) stores the index of data structure `PartialForce`. For example, `ForceIndex[e][v]` indicates where the forces applied on vertex `v` involved by the element `e` are stored in `PartialForce`.

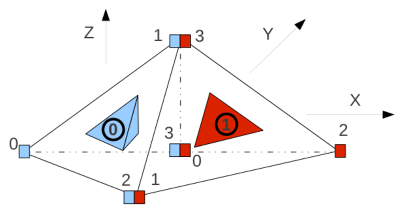
Finally, Algorithm 2 presents the parallel algorithm for force computation (SOFA function `addForce()`). In this algorithm, function `Force()` enables the force computation $F_E(P_i)$ of each node P_i in an element. It depends on constant values (Lamé coefficients λ, ν; coefficients of interpolation functions α_j, β_j for $j = 0..3$) and some variables (initial and current positions of nodes).

Algorithm 2 addForce() function on the GPU

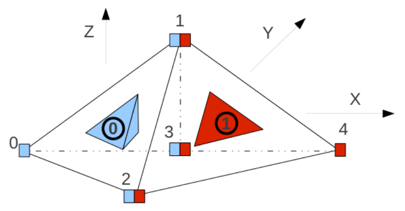
```
1:  {$N$ : number of elements}
2:  {$n$ : number of nodes per element}
3:  {$m$ : total number of nodes}
4:  {$N_n$ : max number of neighbor elements for a node}
5:  // Task 1- Computation of partial forces
6:  for e = 0 to $N-1$ do
7:      // Execution of $N$ kernel1
8:      for v = 0 to $n-1$ do
9:          PartialForce[ForceIndex[e][v]][index[e][v]]=Force(...);
10:     end for
11: end for
12: // Task 2 - Sum of partial forces
13: for i = 0 to $m$ do
14:     // Execution of $m$ kernel2
15:     for j = 0 to $N_n-1$ do
16:         TotalForce[i] += PartialForce[i][j];
17:     end for
18: end for
```

4.2. Illustration

To illustrate this parallelization, consider a deformable object divided into 2 tetrahedra ($N = 2, n = 4, m = 5, N_n = 2$). Fig. 1 points out the local and global indexations of nodes. Fig. 2 presents the data structures filled for our example.



(a) Tetrahedra with local vertex indexation.



(b) Tetrahedra with global vertex indexation.

Figure 1: *Illustration of the parallel computation of forces considering an object discretized into* 2 *tetrahedral elements* ($N = 2, n = 4, m = 5, N_n = 2$).

Consider node P_2 of element E_0, and node P_1 of element E_1. We have `index[0][2]` = `index[1][1]` = 2. Consequently, node P_2 in global indexation is a common vertex of elements E_0 and E_1. Then, we have to compute the partial forces applied on this node involved by these two elements. Considering element E_0 (*resp.* E_1), `ForceIndex[0][2]` = 0 (*resp.* `ForceIndex[1][1]` = 1) indicates where the partial forces computation involved by element E_0 (*resp.* E_1) is stored in `PartialForce`. Consequently, `PartialForce[0][2]` (*resp.* `PartialForce[1][2]`) gives the partial forces computation involved by element E_0 (*resp.* E_1). Then, the sum of these 2 contributions, stored in `TotalForce[2]`, gives the total force applied on node P_2.

	0	1	2	3
0	0	1	**2**	3
1	3	**2**	4	1

(a) `index`.

	0	1	2	3
0	0	0	**0**	0
1	1	**1**	0	1

(b) `ForceIndex`.

	0	1	2	3	4
0	-2	2	**3**	2	2
1		4	**-1**	1	

(c) `PartialForce`.

0	1	2	3	4
-2	6	**2**	3	2

(d) `TotalForce`.

Figure 2: *Data structure for the GPU forces computation.*

4.3. Implicit integration method on the GPU

In section 3.5, we have seen that the use of Euler's implicit integration scheme requires computing the derivatives forces and solving a linear system.

The parallel algorithm of the computation of the derivative forces (SOFA's function `addDForce()`) is similar to Algorithm 2 of the parallel computation of forces. The only difference resides in the use of function `DForce()` instead of `Force()`, which depends on the same parameters but also on the time step h and the current nodes velocity, to directly compute $h^2 \frac{\partial F}{\partial U} V(t)$ or $h \frac{\partial F}{\partial U} \Delta V$, and consequently avoid the storage of the sparse matrix $\frac{\partial F}{\partial U}$.

For the resolution of the system, we used the Conjugate Gradient method and implemented it on the GPU into the framework SOFA, with the same parallel strategy as [ACF11], but using the OpenCL language instead of CUDA.

5. Results and performances

We compare running times on CPU and GPU, using Linux Ubuntu 11.04. The CPU is an Intel® Xeon® CPU 4 cores @3.07 GHz. The GPU is a GeForce GTX 560, 2047 MB, 56 cores @1.620 GHz.

Fig. 3 presents the speedup (ratio between GPU and CPU execution time) obtained considering tetrahedral and triangular elements for the Tensor-Mass model based on linear or non-linear mechanical behaviors (Fig. 4 - 5). Moreover, similar results are presented for SOFA's FEM implementation: it corresponds to the corotational FEM model proposed by

Nesme [NP05] for tetrahedral elements, that considers non-linear behaviors by computing displacements in a rotated local coordinate system. Its GPU version is implemented according to [ACF11]. We obtained a speedup of 25.5 for SOFA's FEM implementation and 29.5 for our TM, for a beam composed of 307,200 elements.

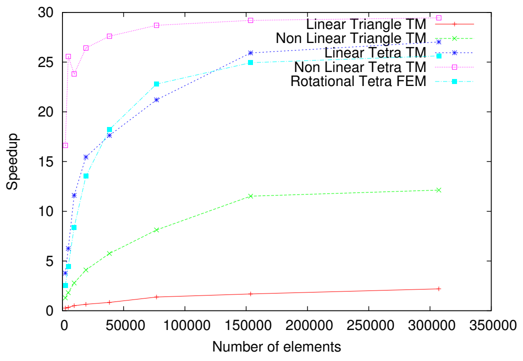


Figure 3: *Speedup between CPU and GPU obtained for the TM model (for tetrahedral and triangular elements) and the FEM implemented into SOFA (for tetrahedral elements).*

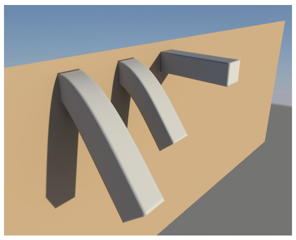


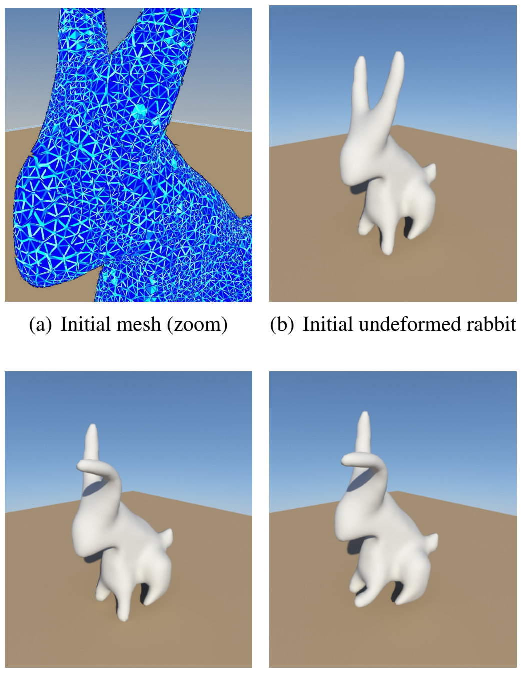
Figure 4: *Rendered beam for Hookean and Saint Venant-Kirchhoff's material, and its initial state (from left to right).*

6. Conclusion and perspectives

In this paper, we presented an original implementation of the Tensor-Mass model on the GPU that considerably speeds up the simulation times. Comparisons between running times on the CPU and the GPU suggest that the parallel implementation of the model becomes interesting for increasingly complex computations.

Besides, we used a formal encoding for the equation forces (function `Force()`). This allows us to generalize this technique to almost any existing combination of constitutive law and mesh type, without extensive additional coding. This is particularly interesting, as it will permit to easily implement and perform a lot of tests and comparisons for various materials on different solvers for an object within the same framework (SOFA, for example). Moreover, the derivatives of the forces may also be obtained formally, which provides an easy way to implement Euler's implicit integration scheme that requires the derivative. This computation would otherwise be very fastidious, as the energy equation must be derived twice; and finally, this solver was never implemented before for the Tensor-Mass model.

In addition, results have been presented both for triangular and tetrahedral meshes, but the derivation for other element types is very easy, as long as interpolation functions Λ exist for this element (for example: quadrangle, quadratic or cubic tetrahedron or hexahedron, prism, etc.).



(a) Initial mesh (zoom) (b) Initial undeformed rabbit

(c) Curved left ear (d) Little jump

Figure 5: *Simulation of the deformation of a rabbit (Initial 3D mesh courtesy of* L. STANCULESCU*).*

References

[ACF*07] 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 *MMVR'15* (Long Beach, USA, February 2007).

[ACF11] ALLARD J., COURTECUISSE H., FAURE F.: Implicit FEM Solver on GPU for Interactive Deformation Simulation. In *GPU Computing Gems Jade Edition*. NVIDIA/Elsevier, Sept. 2011, ch. 21.

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proc. of the 25th annual conference on Computer Graphics and Interactive Techniques* (1998), ACM, pp. 43–54.

[CDA00] COTIN S., DELINGETTE H., AYACHE N.: A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation. *The Visual Computer 16*, 8 (2000), 437–452.

[CTAO08] COMAS O., TAYLOR Z., ALLARD J., OURSELIN S.: Efficient Nonlinear FEM for Soft Tissue Modelling and Its GPU

Implementation within the Open Source Framework SOFA. In *ISBMS 2008, London, UK, July 7-8, 2008: proceedings* (2008), vol. 5104, Springer-Verlag New York Inc, p. 28.

[GW05] GEORGII J., WESTERMANN R.: Mass-spring systems on the GPU. *Simulation Modelling Practice and Theory 13*, 8 (2005), 693–702.

[NMK*06] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically Based Deformable Models in Computer Graphics. *Computer Graphics Forum 25*, 4 (Dec. 2006), 809–836.

[NP05] NESME M., PAYAN Y.: Efficient, physically plausible finite elements. *Eurographics (short papers)* (2005), 1–4.

[PDA00] PICINBONO G., DELINGETTE H., AYACHE N.: Real-Time Large Displacement Elasticity for Surgery Simulation: Non-linear Tensor-Mass Model. In *Proceedings of MICCAI'00* (London, UK, 2000), Springer-Verlag, pp. 643–652.

[Pic03] PICINBONO G.: Non-linear anisotropic elasticity for real-time surgery simulation. *Graphical Models 65*, 5 (Sept. 2003), 305–321.

[RNSS*06] RODRIGUEZ-NAVARRO J., SUSÍN SÁNCHEZ A., ET AL.: Non structured meshes for Cloth GPU simulation using FEM. In *VriPhys 2006* (2006).

[SDR*05] SCHWARTZ J., DENNINGER M., RANCOURT D., MOISAN C., LAURENDEAU D.: Modelling liver tissue properties using a non-linear visco-elastic model for surgery simulation. *Medical Image Analysis 9*, 2 (2005), 103–112.

[SGS10] STONE J., GOHARA D., SHI G.: Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering 12*, 3 (2010), 66.