Implicit Tensor-Mass solver on the GPU

X. Faure, F. Zara, Fabrice Jaillet, J.-M. Moreau

Implicit Tensor-Mass solver on the GPU

X. Faure[1,2] and F. Zara[2] and F. Jaillet[2,3] and J-M. Moreau[2]

[1]Financed by the PRRH (Rhône-Alpes Research Program on Hadrontherapy) for ETOILE (National French Hadrontherapy Centre)
[2]Université de Lyon, CNRS, Université Lyon 1, LIRIS, SAARA team, UMR5205, F-69622, Villeurbanne
[3]Université de Lyon, IUT Lyon 1, Computer Science Department, F-01000, Bourg-en-Bresse

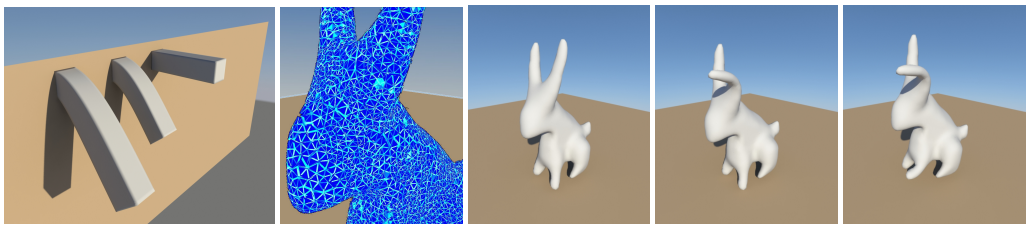

Figure 1: *(1) Rendered beam for Hooke's and Saint Venant-Kirchhoff's material, and initial state (from left to right) - (2-4) Deformation simulation of a rabbit (initial 3D mesh courtesy of L. Stanculescu).*

Abstract

The realist and interactive simulation of deformable objects has become a challenge in Computer Graphics. For this, the Tensor-Mass model is a good candidate: it enables local solving of mechanical equations, making it easier to control deformations from collisions or tool interaction. In this paper, a GPU implementation is presented for the implicit integration scheme. Results show a notable speedup, especially for complex scenes.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry, Object Modeling—Physically based modeling I.6.8 [Simulation And Modeling]: Types of Simulation—Parallel

1. Introduction

The Tensor-Mass (TM) approach is a good candidate to handle deformable objects within the context of physically-based simulations in complex interactive scenes. 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 (cutting) and external interactions (collisions, tools), in the same way as Mass-Spring Systems.

Hence, several formulations have been proposed to account for various mechanical behaviors: the linear Hookean model [CDA00]; the non-linear geometrical model based on Saint Venant-Kirchhoff's elasticity model [PDA00], or anisotropic material [Pic03]; or an extension with pre-computation for non-linear visco-elastic deformations [SDR*05].

But, as far as we know, the problem of the parallel implementation of the TM model has not yet been addressed. In this paper, a GPU implementation is presented, considering both linear and non-linear mechanical behaviors and using an implicit integration scheme to ensure the unconditional stability of our simulation.

2. Simulation of the deformation of an object

The TM approach is based on the domain's discretization into several elements, and the mechanical 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 strain-tensor according to the chosen elasticity model amongst the most employed in interactive simulations, namely a) Hooke's or b) Saint Venant-Kirchhoff's elasticity models -within the element, at X:

  a) $\varepsilon_l(X) = \frac{1}{2}(\nabla U^T(X) + \nabla U(X))$
  b) $\varepsilon_{nl}(X) = \frac{1}{2}(\nabla U^T(X) + \nabla U(X) + \nabla U^T(X)\,\nabla U(X))$

- Computation of the deformation energy W_E, according to the displacement of the element's node;

- Computation of the elasticity force F_E by the evaluation of the derivative of the deformation energy W_E.

3. Force computation on the GPU

The parallel computation of the forces on the GPU is divided into 2 tasks, involving a set of kernels (Alg. 1):

1. First, we compute and store the forces applied on each node of a given 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.

Algorithm 1 Force computation on the GPU

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

Moreover, specific data structures are defined for a discretization of the domain into N elements involving m nodes:

- `index` (of size $N \times n$) stores the relationship between local and global indexation for each node of each element.
- `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.
- `TotalForce` (of size $m \times 3$) stores the sum of the partial forces for each node considering its global indexation.
- `ForceIndex` (of size $N \times n$) stores the index of data structure `PartialForce`.

4. Implicit integration method on the GPU

Then, the dynamical equations are derived from Newton's laws to compute the object acceleration. An implicit integration scheme, requiring computation of the derivatives of forces, is then used to obtain deformation and displacement. The parallel algorithm for this is similar to Alg. 1 - with the only difference in the use of `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 avoid the storage of the sparse Jacobian matrix $\frac{\partial F}{\partial U}$. Next, the linear system is solved on the GPU with the Conjugate Gradient method, following [ACF11].

5. Results and performances

We compare running times on the CPU and GPU, using Linux Ubuntu11.04. The CPU is an Intel® Xeon® 4 cores @3.07 GHz. The GPU is a GeForce GTX 560, 2047 MB, 56 cores @1.620 GHz. Fig. 2 presents the speedup (ratio between GPU and CPU execution time) obtained for triangles and tetrahedra for both the linear and non-linear TM model (Fig. 1). Moreover, similar results are presented for SOFA's FEM implementation, *i.e.* the corotational FEM model for tetrahedra [NP05], in its GPU version according to [ACF11]. A speedup of 25.5 is reached for SOFA's FEM implementation and of 29.5 for our TM, for a beam composed of 307,200 elements.

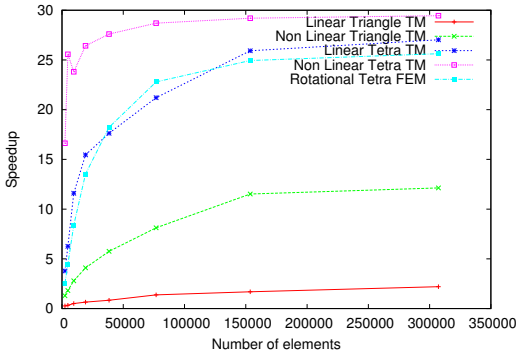

Figure 2: *Performances of the TM parallelization.*

6. Conclusion and perspectives

In this paper, we presented an original implementation of the TM 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.

References

[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. 2

[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. 1

[NP05] NESME M., PAYAN Y.: Efficient, physically plausible finite elements. *Eurographics (short papers)* (2005), 1–4. 2

[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. 1

[Pic03] PICINBONO G.: Non-linear anisotropic elasticity for real-time surgery simulation. *Graphical Models 65*, 5 (Sept. 2003), 305–321. 1

[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. 1