View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by DSpace at University of West Bohemia

A generic topological framework for physical simulation

Elsa Fléchon[a], Florence Zara[a], Guillaume Damiand[a], Fabrice Jaillet[a,b]
[a]Université de Lyon, CNRS, Université Lyon 1, LIRIS, UMR5205, F-69100, Villeurbanne, France
[b]Université de Lyon, IUT Lyon 1, Computer Science Department, F-01000, Bourg-en-Bresse, France
firstname.name@liris.cnrs.fr

ABSTRACT

This paper presents the use of a topological model to simulate a soft body deformation based on a Mass-Spring System. We provide a generic framework which can integrate any kind of geometrical meshes (hexahedral or tetrahedral elements), using several numerical integration schemes (Euler semi-implicit or implicit). This framework naturally allows topological changes in the simulated object during the animation. Our model is based on the 3D Linear Cell Complex topological model (itself based on a 3D combinatorial map), adding the extra information required for simulation purposes. Moreover, we present some adaptations performed on this data structure to fit our simulation requirements, and to allow efficient cutting or piercing in a 3D object.

Keywords

Physically-based simulation; Mass-Spring System; Topological model; Linear Cell Complex; Hexahedral and tetrahedral elements mesh; Deformation; Topological changes; Cutting; Piercing.

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 medical simulation, interactive entertainment, and more generally in all virtual reality applications where animation, interaction or alteration of deformable objects is required in interactive time. Two perennial challenges in this domain are real-time simulation of object undergoing user's interaction like cutting, tearing or fracture, and the adaptive mesh coarsening and refinement, to better handle interaction within a simulation scene (for example in collision and contact zones).

The use of a topological model naturally provides a framework to describe objects subdivided into cells (vertices, edges, faces, volumes) and to modify their topology by performing the appropriate operations. In that case, the information required by the simulation has to be associated with cells. Moreover, an efficient implementation of this kind of data structure should minimize the impact on computation time and should still enable real-time user interactions during the simulation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

The aim of this paper is to put forward the benefits of the use of a topological model for a physical animation based on a Mass-Spring System (denoted MSS). In this sense, we use the 3D Linear Cell Complex (denoted LCC) as topological model (itself based on a 3D combinatorial map). We add on the LCC all information necessary for simulation purposes. Furthermore, we present the adaptations operated on this data structure to facilitate topology changes during the animation, and specifically to allow cutting or piercing of the simulated 3D object. This process is illustrated in Fig. 1.

The use of a topological model for a physical animation presents two main interests. (1) It describes all the cells and all the adjacent and incident relationships between these cells. This is particularly important to associate information to some cells, and to efficiently update this information during the operations. (2) It proposes rigorous operations allowing topological changes while guaranteeing the validity of the mesh. These two features make the topological model unavoidable for a problematic of adaptive mesh refinement.

In this paper, our main contributions are:

- a fully generic framework for topology-based modeling, not only to describe the relationships between geometrical elements, but specifically fitted to the context of physically-based simulation;
- an embedded structure dedicated to storing the mechanical properties, leading to facilitating all the topological changes during animation while preserving the mechanical behavior of the altered object;
- a stable and robust implementation, as the optimized structure will minimize the memory usage and at the

 ISBN 978-80-86943-74-9

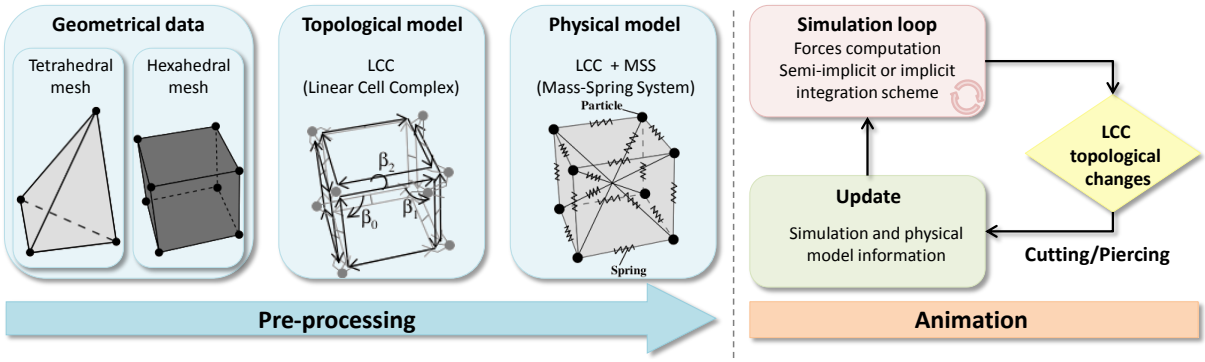


Figure 1: Overview of our method.

same time will permit a simplification of the algorithmic aspects of manipulating elements, *i.e.* low cost browsing through all elements, or accessing to neighbors;

- the detailed formulation of the force differentials (required for the Euler implicit integration scheme) in the context of soft body simulation with MSSs.

2 RELATED WORK

Topological model. Several topological models have been featured in recent years, but only a few of the proposed solutions are fully generic. The combinatorial map is such a solution [1]. It consists of a combinatorial data structure allowing a description of nD objects subdivided into cells (vertices, edges, faces, volumes) using only one basic element, called *dart*, and a set of pointers between these darts. Thanks to these pointers, all the incidence and adjacency relationships between the cells of the subdivision may be easily retrieved.

The main interest of combinatorial maps is: (1) to be generic *i.e.* defined in any dimension; (2) to fully describe the incidence and adjacency relationships between cells, which information is useful for algorithmic aspects; (3) to be possibly customized by adding any type of information to cells. For all these reasons, these models and their variants have already proven to be useful and efficient data structures for image representation and processing [2], or for geometrical modeling [3].

Physical simulation and cutting. In Computer Graphics, significant efforts have been put in proposing efficient methods to model deformable objects, as stated in the following state of the art [4]. Among them, the Finite Element Method (denoted FEM) is the most common. However, it generally requires expensive precomputation to allow interactive topological changes. In [5], a cutting method based on a co-rotational implicit FEM [6] is presented to overcome this limitation, by successively removing, subdividing and adding elements. This avoids the reconstruction of the global matrix, but may generate ill-conditioned elements that are prone to produce numerical instabilities.

The MSS is an interesting alternative for physical modeling. Indeed, as for the tensor-mass model, it supports modification of its geometry in a more natural way, that can be managed locally. Moreover, the principle of splitting faces, instead of removing elements, will result in more plausible animations. The difficulty is then to redistribute the mass and physical parameters over the elements, but none of the past attempts was able to preserve exactly the same behavior as the original MSS. Concerning cutting, some process have been proposed to follow a cutting path either by refining the element or by moving artfully the vertices [7, 8]. This generally implies complex topological modifications and an exhaustive knowledge on the incident and adjacency relationships. For example, an algorithm is proposed in [9] to guarantee the manifold property of the object after topological changes to avoid ill-structured elements. However, this solution requires to handle substantial number of cases. Hence, in our paper, we show a cutting method along edges, particularly tailored for simulation of deformable objects.

Hybrid model. The use of a topological model to dynamically handle changes of object during a physical simulation (generally for cutting purposes) is rare enough to be worthy of note. In 2010, Meseure [10, 11] have presented in this sense a physical simulation based on a MSS using generalized maps, a variant of combinatorial maps. In 2011, Darles [12] has also used the generalized map topological model, this time for simulation based on a model of mass-interaction.

In the first two papers [10, 11], the mechanical information of the objects discretized into tetrahedra is embedded in the topological model by attaching it to the darts. Then, a semi-implicit integration scheme is used for the simulation. But some drawbacks remain in this proposition. Springs are only associated with edges of the topological model. Thus, this cannot be generalized to others geometries, as it will be impossible for example, to add inner diagonal springs in a hexahedron. There is no direct access to particles and springs. Consequently, the topological data structure is computationally expen-

 ISBN 978-80-86943-74-9

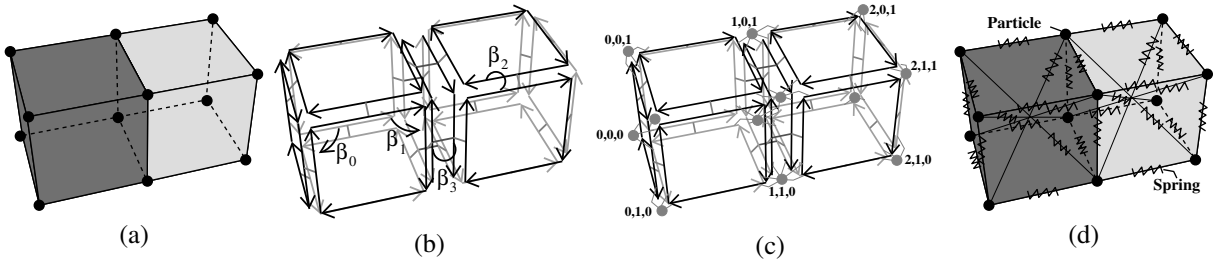


Figure 2: (a) A 3D object composed of two hexahedra (mesh). (b) The corresponding 3D combinatorial map. (c) We add 3D points linked with 0-cell to obtain a 3D LCC. (d) We associate the structures of `Particle` and `Spring` to obtain our LCC+MSS, a 3D LCC allowing a physical simulation based on a MSS.

sive just to retrieve these elements. Thus, they propose to set an additional array containing all the elements plus a pointer through the corresponding dart. This solution improves the speed up of the method, but it leads to a more complex structure and burdens significantly updating operations, as these arrays must be recomputed after each topological modification made on the generalized maps.

The solution introduced in our paper is based on similar ideas as in [10, 11], overcoming the above-mentioned limitations thanks to a more generic model (not only tetrahedra but any type of cells may be taken into account) allowing fully integrated physical simulation (currently based on a MSS) with incremental topological changes. Lastly, high level C++ mechanisms present in CGAL LCC are used. For example, functors are automatically called when a particle is split, allowing us to simplify the updating of mechanical information associated with the topological model.

3 3D LCC FOR MSS

MSSs have largely been used in animation. It consists of discretizing the object in a set of particles (also called masses) connected together by springs. The data structure of our MSS simulation is based on the 3D LCC [13] from the CGAL Open Source geometric algorithms library [14]. This structure allows a representation of an orientable 3D object subdivided into cells with linear geometry.

Fig. 2 illustrates the main steps to construct a 3D LCC for a MSS (denoted LCC+MSS): given the geometrical input data of the 3D object - Fig. 2(a), we first describe its topology with a 3D combinatorial map - Fig. 2(b) which describes cells as well as their incidence and adjacency relationships. Then, we add coordinates of the points to obtain a 3D LCC - Fig. 2(c) which describes the geometry of the objects. Finally, we associate the structure `Particle` with vertices and `Spring` with edges to fit the simulation requirements for a MSS - Fig. 2(d).

3D combinatorial map. We give here an intuitive presentation of combinatorial maps. The interested reader may find all the mathematical background in [1, 15].

In practice, a 3D combinatorial map is an edge-centered data structure composed of a set of basic elements called *darts*, four *pointers* between these darts noted β_0, β_1, β_2 and β_3, and some constraints defined on these pointers to guarantee the topological validity of the described objects.

Fig. 2(a) presents a 3D object composed of 2 3-cells (volumes), 11 2-cells (faces), 20 1-cells (edges) and 12 points associated with the 12 0-cells (vertices). Fig. 2(b) shows the 3D combinatorial map describing this object. Each cell is described by a set of darts. Indeed, each dart (drawn by oriented segments) of the 3D combinatorial maps belongs to a vertex, an edge, a face and a volume. Each cube is described by 24 darts. Given a dart d, $\beta_1(d)$ gives the next dart belonging to the same face and volume and $\beta_0(d)$ gives the previous one; $\beta_2(d)$ gives the other dart belonging to the same edge and volume but not to the same face and $\beta_3(d)$ gives the other dart belonging to the same face and edge but not to the same volume.

Thanks to these rules, starting from a dart, the different pointers can be used to retrieve all the darts that describe the same i-cell, $\forall i \in \{0,1,2,3\}$. Moreover, the adjacency and incidence relationships between the cells are entirely given by these pointers.

This basic description (darts, pointers and constraints) defines a 3D combinatorial map carrying no additional knowledge. But, some information is generally required to describe the geometry of the objects, their colors, the area of their faces, etc. Consequently, in 3D combinatorial maps, any type of information may be associated with any cell. This is done through an association between all the darts that belong to a same cell and an *attribute* containing the information. This allows a direct access to every attribute of a given dart.

3D LCC. The 3D LCC uses the mechanism of attributes to associate a 3D point with each vertex of the combinatorial map (represented by grey dots in Fig. 2(c)). Thus, the geometry of each edge of a 3D LCC is a segment whose endpoints are associated with the two vertices of the edge; the geometry of each face is obtained from all the segments associated with the edges describing the boundary of the facet; and so on.

 ISBN 978-80-86943-74-9

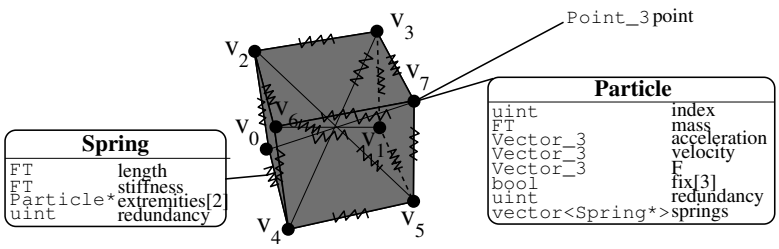


Figure 3: A 3D LCC with additional information of a physical model based on a MSS.

One strength of the CGAL implementation of the LCC is to store every attributes in a compact container which is a set of arrays. Thus, we can directly iterate through all the attributes associated with a given dimension without iterating through darts. Moreover, each array can be processed in parallel, as they are independent. Lastly, the ability to automatically call a function (defined by the user), when two attributes are merged or when an attribute is split in two during an operation, is a powerful mechanism that we use to define our cutting operation (explained in the following).

Another particularly interesting feature is that different geometrical kernels are provided within CGAL, allowing for example the use of exact or inexact arithmetic, exact or inexact geometric predicates. Each kernel defines basic types and geometric primitives. `FT` is the field number type used as basic type to define each coordinate of points and related measures. Depending on the kernel, `FT` could be `double` or an arbitrary precision number type based on the GMP library [16]. Using `FT`, `Point_3` describes a 3D point, and `Vector_3` a 3D vector. As LCC is templated by a kernel, all our code is generic and any type of kernel may be chosen without having to modify anything other than the template argument. This will allow us to easily perform tests of robustness thanks to exact arithmetic to detect possible errors coming from roundness problems.

3D LCC+MSS. To understand our topological model for a MSS, we first briefly recall here the dynamics of a MSS. Thus, considering a MSS composed of n 3D particles, we note $\mathbf{M}$ the diagonal mass matrix (of size $3n \times 3n$), $\mathbf{F}$, $\mathbf{V}$, $\mathbf{P}$ respectively the force, velocity and position vectors of particles (of size $3n$). For each particle i, we note m_i its mass, $\mathbf{P}_i(t), \mathbf{V}_i(t)$ its position and velocity at time t, and $\mathbf{F}_i(t)$ the sum of forces it undergoes (spring forces and external forces like gravity or interaction). For each spring connecting particles i and j, we note k_{ij} its stiffness constant and l_{ij} its initial length.

Then, the dynamics of the model are governed by Newton's law with the following relation at time t for each particle i:

$$m_i \frac{d^2}{dt^2}\mathbf{P}_i(t) = \mathbf{F}_i(t). \quad (1)$$

From this equation, the acceleration of the particles may be deduced according to applied forces. A numerical integration scheme is then used to obtain velocity (according to acceleration) and position (according to velocity) of the particles.

In the current version of our framework, 3D objects are discretized either into hexahedra or tetrahedra. For a tetrahedral discretization, a particle is associated with each vertex, and respectively a spring with each edge of the object. In a same way, for a hexahedral discretization (see example in Fig. 2(d)), a particle is associated with each vertex, a spring with each edge, but in addition there are 4 internal diagonal springs for each hexahedron (with no corresponding edges, 1-cell).

Fig. 3 illustrates with more details the use of a 3D LCC to describe a MSS containing one hexahedron. It contains 8 particles (numbered from v_0 to v_7), 16 springs corresponding to the 12 edges and the 4 internal diagonals of the hexahedron.

For each 0-cell, we store the structure `Particle` corresponding to the information related to the MSS: its index (used for the Euler implicit integration scheme explained later), its mass, its acceleration, its velocity, the sum of the forces applied on this particle, and its direction constraints (to fix the position of a particle along the different axis). We also store the redundancy of the considered particle (the number of volumes incident to the vertex), and the diagonal springs linked to the considered particle, as an array of pointers to `Spring` (only for hexahedral element). Note that the position of the particle is given by the `Point_3` associated to the corresponding vertex in the 3D LCC.

For each 1-cell, we store the associated `Spring` of the MSS: its initial length, its stiffness and an array of pointers to its two extremities (*i.e.* pointers on the two `Particle` connected by the considered spring). We also store information concerning its redundancy (the number of volumes incident to the edge).

Lastly, we store the global stiffness matrix (used for the Euler implicit scheme) into the 3D LCC+MSS object as this matrix is shared by all the volumes of the same 3D LCC.

 ISBN 978-80-86943-74-9

4 TOPOLOGY-BASED SIMULATION

Fig. 1 presents an overview of our method which proposes a topology-based model embedding geometrical and physical information for an efficient response to complex simulation cases. Our method is divided in two parts. (1) A pre-processing part with the construction and the initialization of our model based on the 3D LCC integrating the input parameters of the object (geometrical data, mechanical parameters). (2) An animation part with the simulation loop including the computation of the forces applied on the particles and the numerical integration scheme used to update the velocity and position of the particles; and the possibility of dynamically cutting or piercing our object during the animation. Moreover, we use either the semi-implicit or the implicit version of Euler's integration scheme, leading to fast and stable simulation.

Initialization. The first step of the topology-based simulation consists of the computation of the mechanical information, and in the initialization of the attributes associated with the 3D LCC.

We start by reading an input file describing the geometrical information of the hexahedral or the tetrahedral mesh, and we create the corresponding elementary volumes in the 3D LCC. Then, we identify all the faces that share the same geometry to obtain a connected object.

Next, we iterate through all the particles (0-cells) to initialize the mass, redundancy and index. (1) The index is initialized at the creation of a new particle. (2) The redundancy is the number of volumes incident to the vertex, which can be directly computed using the incidence relations. (3) For the computation of the mass of each particle, the global mass of the object is properly distributed over its n particles. Consequently, considering a 3D homogeneous object discretized into hexahedra with a mass density ρ, the mass m_i of each particle i of the MSS is defined by:

$$m_i = \sum_{j|i \in E_j} \frac{\rho \, V_{E_j}}{8}$$

with E_j the set of hexahedra of volume V_{E_j} containing the particle i. Identically for a tetrahedral discretization, we get:

$$m_i = \sum_{j|i \in E_j} \frac{\rho \, V_{E_j}}{4}$$

with E_j the set of tetrahedra of volume V_{E_j} containing the particle i. Note that it is also possible to attribute a specific mass density to each element, in case of heterogeneous material.

Then, we iterate through all the edges (1-cells) of the LCC to create and initialize the corresponding springs. (1) The stiffness constant is calculated and attributed to each spring of the MSS accordingly to the desired Young modulus E and the Poisson ratio ν (in our examples, we set $E = 100$ MPa and $\nu = 0.3$). Our calculations are based on the formulations given in [17, 18]. (2) The redundancy is the number of volumes incident to the edge, which again can be directly computed. (3) We initialize also the two extremity pointers by using the incidence relationships given by the LCC.

Lastly, we iterate through all the volumes (3-cells) to create diagonal springs when necessary. For each hexahedron, we create the four inner diagonal springs, initialize their information, and insert them into the corresponding particles.

Forces computation. The first step of the simulation's loop is to compute all the forces applied on the particles, due to springs or external interactions. The force involved at time t by a spring connecting particles i and j is defined by:

$$\mathbf{F}_{ij}(t) = \mathbf{F}^e_{ij}(t) + \mathbf{F}^v_{ij}(t) \tag{2}$$

- $\mathbf{F}^e_{ij}(t)$ is the elasticity force of this spring defined by:

$$\begin{cases} \mathbf{F}^e_{ij}(t) & = k_{ij}\left(d_{ij} - l_{ij}\right)\mathbf{U}_{ij}(t) \\ \mathbf{F}^e_{ji}(t) & = -\mathbf{F}^e_{ij}(t) \end{cases}$$

  with $d_{ij} = \|\mathbf{P}_j(t) - \mathbf{P}_i(t)\|$ and $\mathbf{U}_{ij}(t)$ the normalized direction vector defined as:

$$\mathbf{U}_{ij}(t) = \frac{\mathbf{P}_j(t) - \mathbf{P}_i(t)}{\|\mathbf{P}_j(t) - \mathbf{P}_i(t)\|}$$

- $\mathbf{F}^v_{ij}(t)$ is the viscosity force of this spring (used to simulate dissipated energy due to frictions) defined by:

$$\mathbf{F}^v_{ij}(t) = \gamma_{ij}\left[(\mathbf{V}_j(t) - \mathbf{V}_i(t)) \cdot \mathbf{U}_{ij}(t)\right]\mathbf{U}_{ij}(t)$$

  with the spring's viscosity coefficient defined by [19]:

$$\gamma_{ij} = 2\sqrt{\frac{m_i + m_j}{2}\, k_{ij}}$$

To enable this computation with our 3D LCC+MSS model, we first iterate through all the 0-cell attributes to reset the particles force to a null vector, and we add the external forces. In this work we only consider the gravity by adding $m_i\,\mathbf{g}$ to $\mathbf{F}_i$ (with $\mathbf{g} = 9.8$ m/s^2). Then, we iterate through each spring (both non-diagonal and diagonal for hexahedral elements). We compute the force of the considering spring by using equation (2) and we accumulate the calculated force on the two particles linked to it. All the required information is stored in the topological model and may be directly accessed through the attributes (the position, velocity and force of the particles; the extremities, initial length, redundancy, stiffness of the springs).

 ISBN 978-80-86943-74-9

Numerical integration schemes. The second step of the simulation's loop is to compute the velocity and position of all the particles by using a numerical integration scheme. This enables the update of the geometrical coordinates of all the 0-cells of the LCC.

The *Euler semi-implicit* integration method has been implemented first. After the computation of the acceleration of all the particles (using equation (1)), we get for a time step h:

$$\begin{cases} \frac{d}{dt}\mathbf{P}_i(t+h) &= \frac{d}{dt}\mathbf{P}_i(t) + h\frac{d^2}{dt^2}\mathbf{P}_i(t) \\ \mathbf{P}_i(t+h) &= \mathbf{P}_i(t) + h\frac{d}{dt}\mathbf{P}_i(t+h) \end{cases}$$

But, to obtain a stable simulation, h has to be reduced, especially when stiffness increases. So, to enable larger time steps, the *Euler implicit scheme* has been implemented:

$$\begin{cases} \frac{d}{dt}\mathbf{P}_i(t+h) &= \frac{d}{dt}\mathbf{P}_i(t) + h\frac{d^2}{dt^2}\mathbf{P}_i(t+h) \\ \mathbf{P}_i(t+h) &= \mathbf{P}_i(t) + h\frac{d}{dt}\mathbf{P}_i(t+h) \end{cases}$$

This scheme may be reformulated as follows [20]:

$$\left(\mathbf{M} - h\frac{\partial \mathbf{F}(t))}{\partial \mathbf{V}(t)} - h^2\frac{\partial \mathbf{F}(t)}{\partial \mathbf{P}(t)}\right)\Delta\mathbf{V} = h\,\mathbf{F}(t) + h^2\frac{\partial \mathbf{F}(t)}{\partial \mathbf{P}(t)}\mathbf{V}(t)$$

with

$$\Delta\mathbf{V} = \frac{d}{dt}\mathbf{P}(t+h) - \frac{d}{dt}\mathbf{P}(t)$$

and $\partial\mathbf{F}/\partial\mathbf{P}$, $\partial\mathbf{F}/\partial\mathbf{V}$, the Jacobian matrices (of size $3n \times 3n$, for n particles) encoding the variation of forces resulting from position and velocity change at time t. After computing these matrices, this linear system is solved using the Conjugate Gradient method to obtain $\Delta\mathbf{V}$. Then, the velocity is updated with:

$$\frac{d}{dt}\mathbf{P}(t+h) = \frac{d}{dt}\mathbf{P}(t) + \Delta\mathbf{V}$$

and the position with:

$$\mathbf{P}(t+h) = \mathbf{P}(t) + h\frac{d}{dt}\mathbf{P}(t+h)$$

Naturally, the damping coming from the environment is taken into account when updating the velocity of the particles. Moreover, we can note that the acceleration is never really computed according to equation (1) within the Euler implicit scheme.

Computation of the global stiffness matrix $\partial\mathbf{F}/\partial\mathbf{P}$**.** To fill, at time t, the nonzero entries of matrix $\partial\mathbf{F}/\partial\mathbf{P}$, we compute $\partial\mathbf{F}_{ij}/\partial\mathbf{P}_i$ (matrix of size 3×3) only if particles i and j are connected (with $i \neq j$), with:

$$\frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{P}_i} = \frac{\partial \mathbf{F}^e_{ij}}{\partial \mathbf{P}_i} + \frac{\partial \mathbf{F}^v_{ij}}{\partial \mathbf{P}_i}$$

with

$$\begin{cases} \frac{\partial \mathbf{F}^e_{ij}}{\partial \mathbf{P}_i} = k_{ij}\left[\frac{l_{ij}}{d_{ij}}(I - \mathbf{U}_{ij}\,\mathbf{U}^T_{ij}) - I\right] \\ \frac{\partial \mathbf{F}^v_{ij}}{\partial \mathbf{P}_i} = \frac{\gamma_{ij}}{d_{ij}}\left[[\mathbf{U}_{ij}\mathbf{U}^T_{ij} - I]\omega + [W\,\mathbf{U}_{ij}\mathbf{U}^T_{ij}]^T\right] \end{cases}$$

where I is the identity matrix of size 3×3 and W a diagonal matrix defined by:

$$W = -\frac{\mathbf{V}_{ij}}{\mathbf{U}_{ij}} + \omega \times (1,1,1)^T \text{ with } \omega = \mathbf{V}_{ij} \cdot \mathbf{U}_{ij}$$

Fig. 4 illustrates the matrix $\partial\mathbf{F}/\partial\mathbf{P}$ for a 2D MSS with 6 particles. This matrix is composed of:

- non-diagonal elements $\partial\mathbf{F}_{ji}/\partial\mathbf{P}_i$ and $\partial\mathbf{F}_{ij}/\partial\mathbf{P}_j$ deduced from $\partial\mathbf{F}_{ij}/\partial\mathbf{P}_i$ with:

$$\frac{\partial \mathbf{F}^{e/v}_{ij}}{\partial \mathbf{P}_i} = -\frac{\partial \mathbf{F}^{e/v}_{ji}}{\partial \mathbf{P}_i}, \frac{\partial \mathbf{F}^{e/v}_{ji}}{\partial \mathbf{P}_i} = \frac{\partial \mathbf{F}^{e/v}_{ij}}{\partial \mathbf{P}_j}$$

- diagonal elements $[\partial\mathbf{F}/\partial\mathbf{P}]_{i,i}$ with:

$$\left[\frac{\partial \mathbf{F}}{\partial \mathbf{P}}\right]_{i,i} = \sum_{k \in S}\frac{\partial \mathbf{F}_{ik}}{\partial \mathbf{P}_i}$$

  with S the set of particles connected to i.

Note that this global stiffness matrix is filled using the particle index.

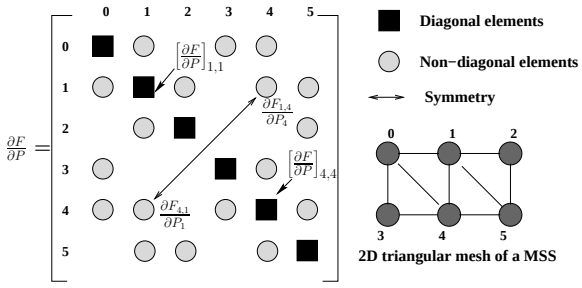


Figure 4: Illustration of the Jacobian matrix $\partial\mathbf{F}/\partial\mathbf{P}$ for a 2D MSS composed of 6 particles.

Computation of the damping matrix $\partial\mathbf{F}/\partial\mathbf{V}$**.** The matrix $\partial\mathbf{F}/\partial\mathbf{V}$ corresponds to the Rayleigh Damping defined by $\mu M + \lambda\,\partial\mathbf{F}/\partial\mathbf{P}$, with μ and λ the mass and stiffness proportional Rayleigh damping coefficients.

5 TOPOLOGICAL CUTTING

Amongst all the changes that a 3D object may undergo during an animation, cutting is one of the most challenging, as it implies deep topological changes, *i.e.* element removing or splitting. Without an efficient topological model, it may cause difficulties, as generating

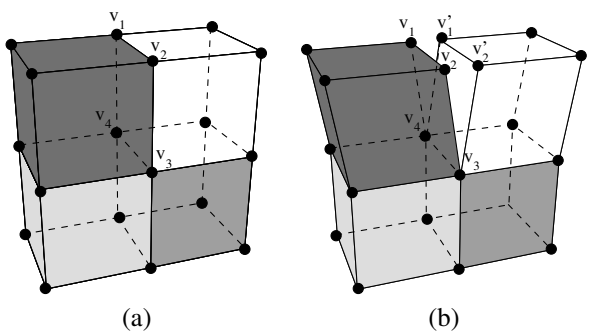
 ISBN 978-80-86943-74-9

ill-structured and non-manifold mesh when refining elements or moving points to follow the cutting path. In this section, we demonstrate through this particular operation that the proposed model is robust and well adapted, with limited additional computational cost.

Unsewing in 3D LCC. In a 3D LCC, the `i-unsew` operation, $i \in \{1,2,3\}$, unglues two i-cells which are glued along one of their $(i-1)$-cells. For that, we unlink β_i pointers for all the darts belonging to the shared $(i-1)$-cell. After this operation, the initial shared $(i-1)$-cell is split in two $(i-1)$-cells, and respectively this initial $(i-1)$-cell will no more be shared by the two i-cells.

If the unsew operation splits a j-cell c, $\forall j \in \{0,1,2,3\}$, in two j-cells $c1$ and $c2$, and if c is associated with a j-attribute `attr1`, then this attribute is duplicated into `attr2`, and all the darts belonging to $c2$ are associated with this new attribute. Next, a functor is called on the two attributes allowing the user to update its specific information.

In Fig. 5, an example of `3-unsew` operation is presented. We start from the LCC given in Fig. 5(a) made up of 4 hexahedra. A `3-unsew` operation is processed to unglue the dark grey and the white hexahedra. The face separating these two hexahedra (named (v_1, v_2, v_3, v_4) in the initial configuration) is split in two by the `3-unsew` operation. We can see in Fig. 5(b) that this split involves the duplication of the two vertices v_1 and v_2, and the duplication of the three edges (v_4, v_1), (v_1, v_2) and (v_2, v_3). The user defined functor is called on each pair of duplicated cells, for example on (v_1, v_1') for vertices, and $((v_4, v_1), (v_4', v_1'))$ for edges. In this example, note that vertices v_3 and v_4 are not duplicated during the unsew operation as they are still connected by the two grey hexahedra right below, and consequently the edge (v_3, v_4) is not duplicated either.

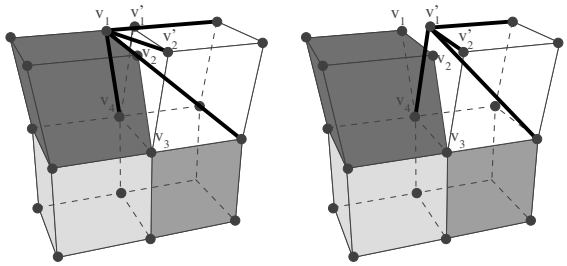


(a) (b)

Figure 5: Example of 3-unsew in a 3D LCC.

Object cutting during simulation. During the physical simulation, we use the `3-unsew` operation to cut an object by separating two adjacent volumes. As explained above, this operation duplicates the attributes where necessary. In our case, particles and springs are possibly duplicated. In both cases, the physical information stored in the attributes has to be updated.

For new particle. If the cutting involves the creation of a new particle (for example particle v_1' in Fig. 5), all the information is first duplicated from the initial particle (v_1). Then we initialize the index of v_1' to a new index (used for the Euler implicit integration) and we update the mass of v_1 and v_1' by decrementing their redundancy (*i.e.* the number of volumes incident to the particles). Lastly, we update the list of diagonal springs connected to v_1 and v_1'. Indeed, springs associated with the second volume are still attached to the initial particle v_1, that is incorrect (see Fig. 6(a)). Thus, we iterate through all the springs associated with v_1 and for each one whose other extremity belongs to the second volume, we detach it from v_1 and attach it to v_1' (see Fig. 6(b)).



(a) (b)

Figure 6: Modification of springs in the 3D LCC+MSS after 3-unsewing. (a) The bold segments represent the springs wrongly attached to particle v_1 before updating. (b) Same springs after updating.

For new spring. If the cutting involves the creation of a new spring (for example edge (v_1', v_2') in Fig. 5), all the information of the spring associated with the initial edge (here, the edge (v_1, v_2)) is duplicated on the new spring created and associated with the new edge. Then, we update the redundancy of the two springs by counting the number of volumes incident to each corresponding edge, and we update the two extremities of the new spring to link the two particles incident to the new edge.

As a consequence of the creation of new particles, the cutting of the object involves the re-sizing of the data structures that store the mechanical information: force, acceleration (for Euler's semi-implicit scheme), velocity, position vectors, and the global stiffness matrix (for Euler's implicit scheme). Then, the computation of the forces applied on each particle is performed as usual.

6 RESULTS

In this section, we present some results to validate the behavior of our animation based on a topological model. We show how our model can simulate deformation of a soft body and how it can be cut or pierced

 ISBN 978-80-86943-74-9

during the animation. All our experiments were carried out on an Intel Core i7 processor (2.40 GHz with 4 multi-threaded cores). We set the material properties to $E = 100$ MPa, $\nu = 0.3$ and $\rho = 1{,}000$ kg/m^3 to simulate behaviors similar to soft tissues. All the given times correspond to one step of the simulation process, in milliseconds. For all the presented results, only the gravity force is applied. Moreover, the red spheres in figures are particles constrained in all directions and the 3-unsewed faces are drawn in red.

Semi-implicit *vs.* implicit integration scheme. We simulate a cube modeled by 1 hexahedron with 16 springs and 8 particles, and the same cube modeled with 5 tetrahedra, 18 springs and 8 particles. Table 1 compares the computation times resulting from the use of the Euler semi-implicit and implicit integration schemes (with $h = 1$ ms). As expected, the implicit one is slower, but this will be advantageously counterbalanced by more stability, allowing larger time steps.

	Semi-implicit	Implicit
Hexahedral mesh	0.009	0.42
Tetrahedral mesh	0.012	0.42

Table 1: Time (in ms) per simulation step.

Scale up. Fig. 7 presents the scale up property of our simulation by considering a beam with an increasing number of hexahedral elements: 1 cube of 10 cm; $2 \times 2 \times 2$ cubes of 5 cm; $4 \times 4 \times 4$ cubes of 2.5 cm; $8 \times 8 \times 8$ cubes of 1.25 cm; $16 \times 16 \times 16$ cubes of 0.625 cm (simulations made with $h = 0.1$ ms) and $32 \times 32 \times 32$ cubes of 0.3125 cm (simulation made with $h = 0.01$ ms). Results show that the complexity of our method is linear according to the degree of freedom (DOF) within our system.

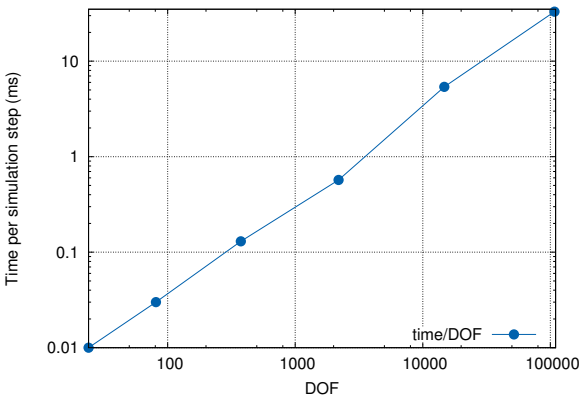


Figure 7: Time (in ms) per simulation step according to the size of a hexahedral beam (with log-log scale).

Comparison performance with SOFA. We now consider a mesh with 640 hexahedra, 4,954 springs and 891 particles. We compare our simulation times with the ones obtained using our same MSS implemented in the Open Source Framework SOFA [21, 22]. With our topological framework, we obtain an average of 0.55 ms by simulation step with the Euler semi-implicit integration scheme, while we get 0.77 ms by using the Euler explicit integration scheme in SOFA (with $h = 1$ ms). We note that the additional cost due to the topological structure remains limited, using a similar integration scheme. This first comparison is very encouraging as it shows that our method is competitive even in its preliminary form (not fully optimized).

Comparison with another topological model. In [11] a similar solution has been proposed. With a mesh composed of 1,856 tetrahedra, 2,850 springs and 564 particles, they stated that the simulation step takes 8 ms on a dual core 2.8GHz processor, using an Euler semi-implicit integration (4th order Runge-Kutta integration scheme).

For comparison purposes, we built a similar beam composed of 1,890 tetrahedra, 2,655 springs and 480 particles. We obtain an average of 0.5 ms by simulation step when using the Euler semi-implicit integration (with $h = 1$ ms). Even if these results do not involve the same CPU, nor exactly the same mesh, this preliminary comparison is very satisfactory and demonstrates that the proposed structure is well adapted for simulation applications, leading to minimizing the additional memory and operative cost induced by the topological model.

Cutting. As presented above, cutting objects during the animation is easily supported by the proposed structure. In the following, the Euler implicit integration scheme is used for its stability when high deformation is undergone by the 3D object.

Fig. 8, 10 and 11 show examples of interactive deformation and cutting of a beam. The user is provided with tools permitting him to select the particles belonging to the face to 3-unsew. If necessary, a zone can be selected by the user, where all the faces including the selected particles are 3-unsew.

In Fig. 9, we present the cutting performed on two bigger meshes representing a frog. This illustrates that our method can be used for detailed objects.

Piercing. Thanks to our cutting method, we can pierce an object by 3-unsewing all the faces around the volume to be removed. In Fig. 12, we pierce a beam composed of $5 \times 3 \times 3$ hexahedral elements by 3-unsewing 3 cubes in the middle of the beam.

7 CONCLUSION AND PERSPECTIVE

In this paper, we have presented the use of the 3D LCC topological model for a physical simulation based on a MSS. The mechanical information of the object is added to the data structure as attributes associated with i-cells, avoiding the duplication of data and the management of different data structures. Our first experiments illustrate that our method is competitive, even without any specific optimization.

 ISBN 978-80-86943-74-9

In the future, we first plan to improve our simulation time. All the operations used for the simulation can be easily implemented in parallel, and a version of the simulation on the GPU is under investigation. Moreover, we work to insert other physical models like tensor-mass model, finite element method or even a more evolved mass-spring system. Finally, we want to integrate in our framework the automatic refining and coarsening of cells during the simulation. Indeed, the 3D LCC provides all the basic topological operations required for that kind of operation. Consequently, the next stage will be: how to update the physical information and to define the criteria to decide where these operations must be applied? This improvement will allow us to overcome the current limitation of cutting only along the faces of the mesh. Indeed thanks to the subdivision features, we will be able to follow a cutting path through elements.

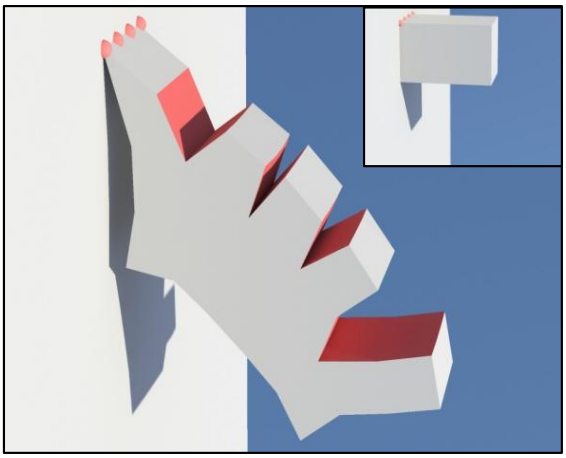


Figure 8: 10 faces are 3-unsewed of a beam composed of $5 \times 3 \times 3$ elements (initial state in top right).

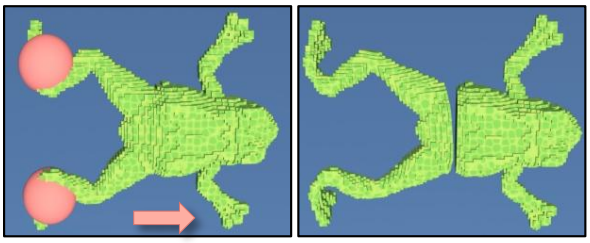


Figure 9: Cutting one hexahedral mesh representing a frog with 26,125 hexahedra and 32,934 particles.

ACKNOWLEDGEMENT

The authors would like to thank Eric Galin for his helpfull comments. This work was performed within the framework of the LabEx PRIMES (ANR-11-LABX-0063) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR); and partially funded by the ANR-MN project SAGA (ANR-12-MONU-0006).

8 REFERENCES

[1] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Comput. Aided Des.*, 23(1):59–82, 1991.

[2] G. Damiand. Topological model for 3D image representation: Definition and incremental extraction algorithm. *Computer Vision and Image Understanding*, 109(3):260–289, March 2008.

[3] D. Fradin, D. Meneveaux, and P. Lienhardt. A hierarchical topology-based model for handling complex indoor scenes. *Computer Graphics Forum*, 25(2):149–162, June 2006.

[4] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[5] H. Courtecuisse, H. Jung, J. Allard, C. Duriez, D. Y. Lee, and S. Cotin. GPU-based real-time soft tissue deformation with cutting and haptic feedback. *Progress in Biophysics and Molecular Biology*, 103(2-3):159–168, 2010. Special Issue on Soft Tissue Modelling.

[6] M. Nesme, Y. Payan, and F. Faure. Efficient, Physically Plausible Finite Elements. In *Eurographics'05 (short papers)*, Dublin (IRL), 2005.

[7] B. Lee, D. C. Popescu, and S. Ourselin. Topology modification for surgical simulation using precomputed finite element models based on linear elasticity. *Progress in Biophysics and Molecular Biology*, 103(2-3):236–251, 2010. Special Issue on Biomechanical Modelling of Soft Tissue Motion.

[8] C. Dick, J. Georgii, and R. Westermann. A hexahedral multigrid approach for simulating cuts in deformable objects. *IEEE TVCG*, 17(11):1663–1675, 2011.

[9] C. Forest, H. Delingette, and N. Ayache. Removing Tetrahedra from manifold tetrahedralisation : application to real-time surgical simulation. *Medical Image Analysis*, 9(2):113–122, 2005.

[10] P. Meseure, E. Darles, and X. Skapin. A Topology-Based Mass/Spring System. In *Proc. of CASA'2010 (short papers)*, St Malo (F), June 2010.

[11] P. Meseure, E. Darles, and X. Skapin. Topology-based Physical Simulation. In *Proc. of VRIPHYS'10*, pages 1–10, Copenhagen (DK), November 2010.

[12] E. Darles, S. Kalantari, X. Skapin, B. Crespin, and A. Luciani. Hybrid physical–topological modeling of physical shapes transformations. In *Proc. of DMDCM'11*, pages 154–157, Washington, DC (USA), 2011.

[13] G. Damiand. Linear Cell Complex. In *CGAL User and Reference Manual*. CGAL Editorial Board. http://www.cgal.org/Pkg/LinearCellComplex.

[14] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.2 edition. http://www.cgal.org/.

[15] G. Damiand. Combinatorial maps. In *CGAL User and Reference Manual*. CGAL Editorial Board. http://www.cgal.org/Pkg/CombinatorialMaps.

 ISBN 978-80-86943-74-9

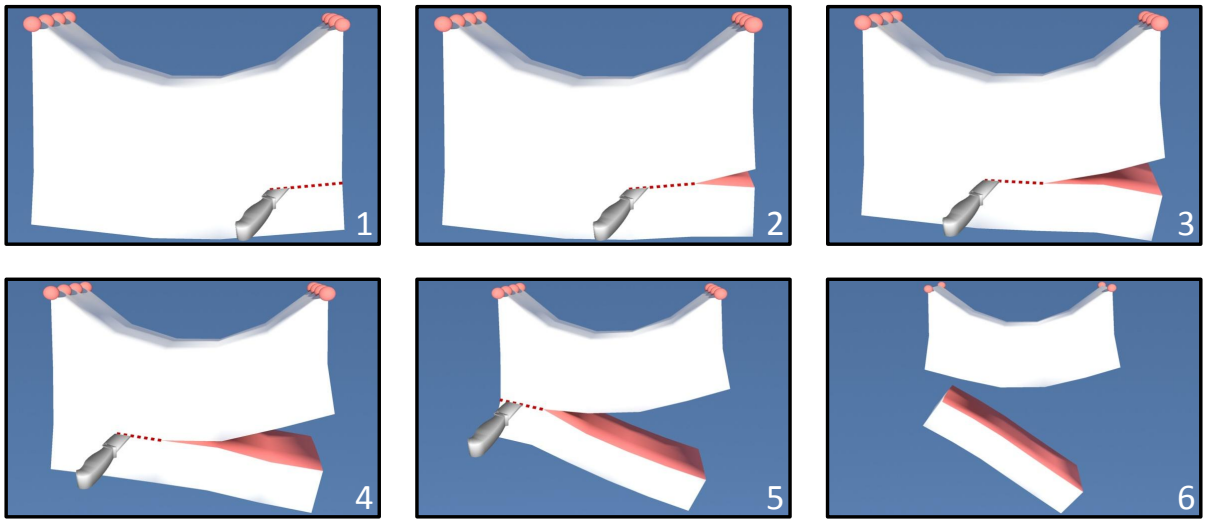


Figure 10: Beam composed of $5 \times 3 \times 3$ elements, progressively cut by a knife (from the Avalon 3D archive).

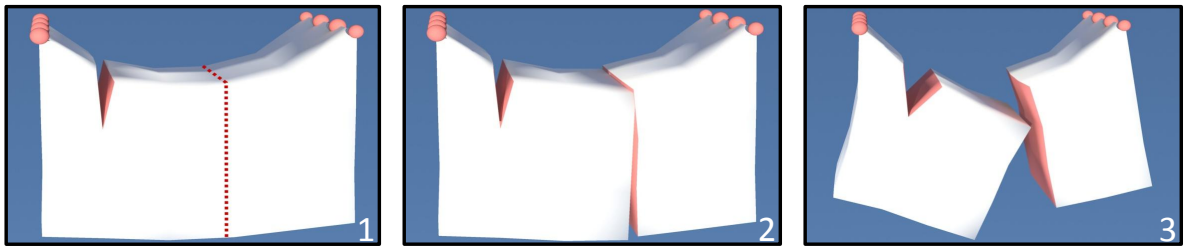


Figure 11: Beam composed of $5 \times 3 \times 3$ elements. 1) 3 faces are 3-unsewed. 2) and 3) Two states after the 3-unsewing of 9 faces.

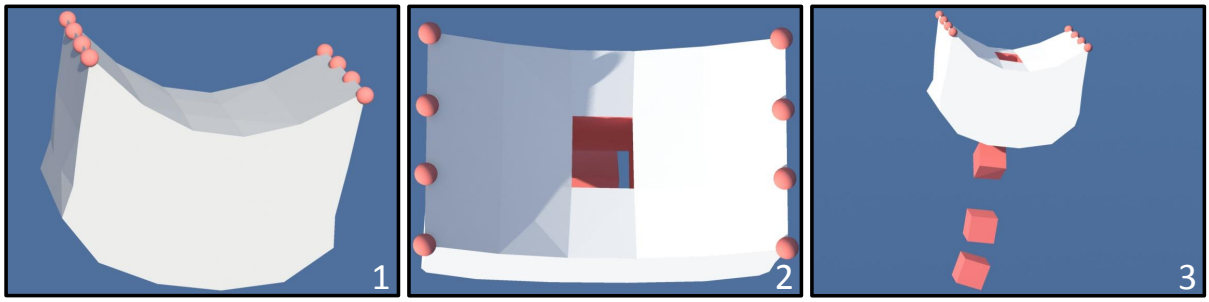


Figure 12: Beam composed of $5 \times 3 \times 3$ hexahedral elements. 1) Initial deformed state. 2) and 3) Two views of the same object after piercing by 3-unsewing 14 faces.

[16] Gmp library : http://gmplib.org.

[17] V. Baudet, M. Beuve, F. Jaillet, B. Shariat, and F. Zara. Integrating Tensile Parameters in Hexahedral Mass-Spring System for Simulation. In *Proc. of WSCG'09*, pages 145–152, February 2009.

[18] V. Baudet, M. Beuve, F. Jaillet, B. Shariat, and F. Zara. Integrating Tensile Parameters in Mass-Spring System for Deformable Object Simulation. Technical report, LIRIS UMR CNRS 5205, September 2009.

[19] Y. Bhasin and A. Liu. Bounds for damping that guarantee stability in mass-spring systems. *Studies in health technology and informatics*, 119:55–60, 2005.

[20] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proc. of the 25th annual conference on Computer Graphics and Interactive Techniques*, pages 43–54. ACM, 1998.

[21] J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni. SOFA - an Open Source Framework for Medical Simulation. In *MMVR 15*, pages 13–18, Palm Beach, USA, 2007.

[22] Sofa : http://www.sofa-framework.org.

 ISBN 978-80-86943-74-9