Topology-based Physical Simulation

Philippe Meseure, Emmanuelle Darles, Xavier Skapin

▶ **To cite this version:**

Philippe Meseure, Emmanuelle Darles, Xavier Skapin. Topology-based Physical Simulation. Virtual Reality Interaction and Physical Simulation 2010 (VRIPHYS), Nov 2010, Copenhague, Denmark. pp.1-10. hal-00563396

HAL Id: hal-00563396
https://hal.science/hal-00563396

Submitted on 24 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2010), pp. 1–10

Topology-based Physical Simulation

Philippe Meseure†, Emmanuelle Darles, Xavier Skapin

University of Poitiers, XLIM-SIC CNRS UMR 6172

Abstract

This paper presents a framework to design mechanical models relying on a topological basis. Whereas naive topological models such as adjacency graphs provide low topological control, the use of efficient topological models such as generalized maps guarantees the quasi-manifold property of the manipulated object: Topological inquiries or changes can be handled robustly and allow the model designer to focus on mechanical aspects. Even if the topology structure is more detailed and consumes more memory, we show that an efficient implementation does not impact computation time and still enables real-time simulation and interaction. We analyze how a simple mass/spring model can be embedded within this framework.

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

Introduction

Various physical models have been proposed in the physically-based animation field during the two last decades. These models often include some topological information dedicated to specific requirements, e. g. identifying the vertices of each volume in finite element methods, or identifying the faces around a vertex to compute a surface normal before visualization.

The use of topological models is usually restricted to the modeling field (any modeler relies on some sort of topological model), but to our knowledge, no physically-based model has ever been proposed which includes general purpose topological information while providing various and adequate modifications. In some given applications, topological information is necessary, especially when topological changes are undergone by the model. In surgical simulation, cuttings, tearings or destructions of tissue are topological operations which are performed during the simulation and not as a preprocessing phase.

This paper aims at giving some clues about how to base a physical model on a convenient topological model, namely, *generalized maps*. The goal of this topological model is definitely not to spend less memory or even speed up the computation. It rather provides robustness, due to rigorous mathematical definitions, and versatility since it is easy to define new operations. Maps provide an adequate help to any neighborhood issue. Moreover, some ill-structured objects (i.e. "non-manifolds") cannot be met after applying a topological modification since they are impossible to represent using maps. We therefore eliminate the risk of creating pending faces or volumes only linked by a vertex or an edge (topological representations of meshes that define connectivity by the edges can not prevent such situations). Furthermore, recent work has shown that collisions and self-collisions can be handled in a robust way using topological tracking [JCD09, LSM08].

Our goal is to show that topological models provide a framework to implement topological transformation in a robust way and avoid topological issues such as incorrect determination of neighbors for instance. However, their use requires that the mechanical model relies on the topology. All the associated algorithmic aspects must be designed subsequently, in terms of coverage of the structure. Thereafter, we need to define the mechanical modifications implied by all the considered topology changes. The main contributions of our work are: (1) a general framework to provide existing mechanical models with a topological basis in order to represent neighborhood relationships explicitly and avoid ill-

† *Philippe.Meseure@sic.univ-poitiers.fr*

submitted to *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2010)*

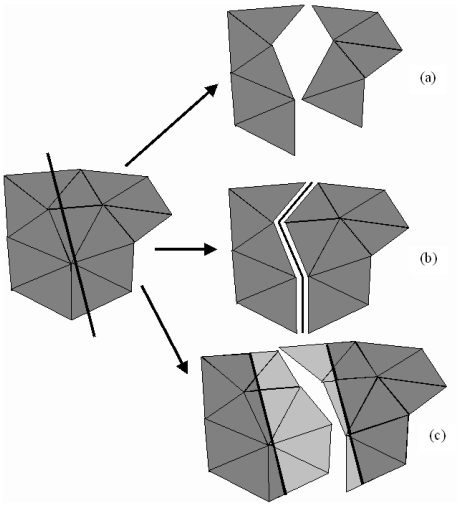

Figure 1: *2D mesh cutting approaches: (a) removing of faces. (b) separation of faces. (c) duplication of faces.*

structured objets, (2) a mass/spring model based on such a framework and (3) a robust approach to handle topological changes in mechanical models.

This paper is organized as follows: After describing previous work, we detail the main principles of topological models and the specific vocabulary related to this field. Then, we show how we can "embed" a topological model with mechanical information. In the following section, we present how we have instantiated such an approach to a mass/spring mesh. Section 5 gives some optimizations to achieve real-time requirements.

1. Previous work

Some early studies on physically-based animation have focused on topological changes to simulate fractures [TF88, NTB*91] or fusion [TPF89, Ton91]. While most of them usually rely on simple mechanical models (particles or rigid body), more recent approaches have applied topological changes to explicit finite-element models [OH99].

Surgical simulation requires to apply such changes in real-time, and several methods have been proposed to simulate cuttings [CDA00, FG99, NvdS00]. The submitted solutions have mainly focused on the mechanical adaptation related to some topological changes, that, in the 3D (resp. 2D) case, fall into one of these three approaches:

- volumes separation (resp. faces): neighborhood relationships are discarded by dissociating the common face, edges and vertices (resp. edge and vertices) between two volumes (resp. faces) [FG99, NvdS00]
- volumes (resp. faces) removal [CDA00]
- volumes (resp. faces) duplication [MBF04, SDF07].

The different approaches in 2D are shown in Fig. 1. They imply different topological changes of various complexities. The complexity of the operation comes both from the topological change itself and the computation of the local adaptation of physical values that usually requires the model to provide neighborhood relations (it can be useful to enumerate volumes adjacent to a vertex to compute its new mass for instance). Separation and removal of volumes are actually complementary and it is convenient to combine both. Separation of volumes is relevant to simulate scalpel cutting whereas volumes removal reproduces destruction or burning of tissue quite well. The topological basis allows us to unify both approaches by relying on the same elementary modifications: face dissociation (in the remainder of this article, we will call "dissociation" the separation of common cells between two separating volumes instead of the more usual "splitting" term that is ambiguous in our context).

Note that when these approaches are used alone, the result is approximate because it heavily relies on the mesh resolution. For instance in surgical simulation, the geometry of splitting usually does not fit the intended cutting path. Geometrical adaptation [NvdS03], division of volumes into multiple sub-volumes [BG00] or mesh local refinement [FDA05] are different approaches that address this issue. In the case of meshless models, an approach relying on co-refinement has also been proposed [SOG06]. These different approaches imply complex topological modifications as well. Although they are beyond the scope of this paper, they would surely benefit from a topologically-robust approach as the one we propose here.

Most approaches are actually based on adjacency graphs and are indexed most of the time. In practice, vertices are numbered in an array providing positions and usually other features like velocities, forces and various mechanical properties. Edges, faces and volumes are defined as an ordered sequence of vertex indices. First, it has been shown that adjacency graphs are ambiguous (several non topologically-equivalent objects can rely on the same adjacency graph) [LFB08]. Second, such models do not explicitly encode all the needed adjacency relationships. Additional information is often required (for instance, a face could also contain a list of its edges, a vertex could enumerate its surrounding volumes, etc.). This information must be updated in a consistent way during topology changes. Nevertheless, since the different steps of the simulation (forces computation, integration of differential equations, visualization) require different neighborhood relations, this results in a very heavy structure whose consistency is hard to maintain. To sum up, relying on naive topological models lead to non robust topological handling algorithms.

Furthermore, as shown by Forest et al. [FDA05], the simulated body should remain manifold during topological changes, in order to avoid ill-structured elements (see Fig. 2). However, instead of using a powerful topological

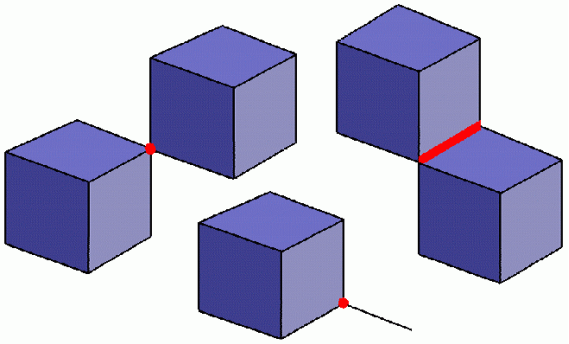

Figure 2: *Examples of non-manifold objects that must be avoided after applying some topological modification.*

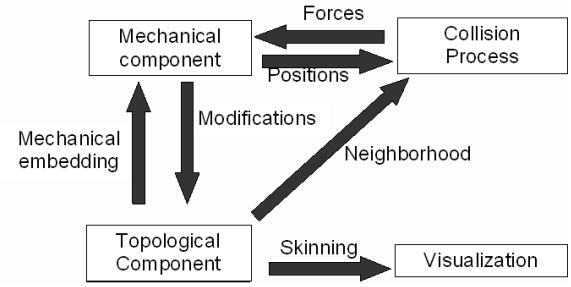

Figure 4: *Architecture of the topologically-based mechanical model.*

structure, they proposed an algorithm to guarantee that the object remains manifold (a *n*-dimensional manifold is defined as a topological space where every point has a neighborhood homeomorphic to a ball). The algorithm is quite heavy, several cases of ill-constructed structures must indeed be checked after any modification. Besides, instead of restricting the objects to manifolds, we prefer to consider a larger class of objects: quasi-manifolds. Quasi-manifolds are defined, in 3D, as a set of volumes which are linked by faces, provided that each face links at most two volumes. Fig. 3 shows an example of a non-manifold object that is a physically computable quasi-manifold.

Obviously, some studies have tried to use more appropriate approaches. Some approaches only focus on the cellular structure of the model itself and the simulation algorithms (for instance, using *chains* on cell complexes [ES00]), but do not consider any topological modification. Other models are based on simplex meshes [DSCP94], but restrict the topological modifications to volume separations. Some approaches have tried to use widely-known topological models such as winged-edges [Bau75]. Nevertheless, these models

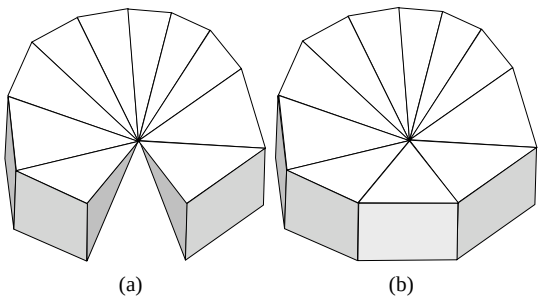

Figure 3: *(a) a manifold object and (b) a quasi-manifold object. Indeed, the neighborhood of the central vertex is not homeomorphic to a ball (but rather a kind of "pinched" ball)*

are restricted to dimension 2 and can only reproduce surface cuttings or tearings.

Considering previous work, we understand that a framework that enables any topological modifications and maintains the model consistency is required. This framework aims at solving topological issues so that a designer can focus on the mechanical adaptation of his model only. This framework implies to build the animation model (usually composed of three parts to compute mechanical evolution, collisions and visualization) on an additional component as shown in Fig. 4. Thus, the topological component appears as a major part that supports at least visualization, behavior and, if possible, collisions.

2. Topological models

The context we have exposed leads us to use, among all topological models, a versatile one, which does not constrain any type of face or volume and enables multiple and complex modeling operations, including refinement for instance. Since the objects we manipulate can be represented as subdivisions, we find that topological maps are rather convenient for our purpose [Lie94]. *Combinatorial maps* [Edm60] are especially relevant, since they allow us to represent the class of objects we manipulate, namely orientable quasi-manifolds. Nevertheless, in this paper, we choose to rely on *generalized maps*. This class of maps can represent orientable or non-orientable quasi-manifolds but they require twice as much memory as combinatorial maps. However, their algebraic definition is homogeneous (in dimension *n*), and, compared to combinatorial maps, we found them easier to present to a reader who is not familiar with topological models. Generalized maps do not significantly change the results obtained with our approach. Furthermore, methods to export generalized maps to combinatorial maps are well-known, so the results presented in this paper can be adapted. Precise relations between generalized, combinatorial maps and other data structures are presented in [Lie91].

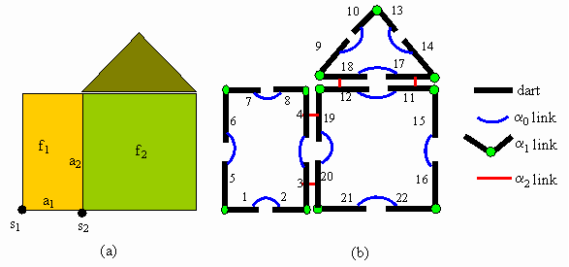

Figure 5: *(a) A 2D subdivision. (b) The corresponding 2-g-map. Darts are represented by numbered black segments.*

2.1. Generalized maps

A n-dimensional generalized map (denoted n-g-map) is based on a unique abstract element, called *dart* that, for 3D-objects, roughly represents "a vertex on an edge of a face of a volume of the object". n-g-maps are defined as a set of darts and applications defined on these darts:

Definition (Generalized map) Let $n \geq 0$. An n-dimensional generalized map (or n-g-map) is defined by $G = (B, \alpha_0, \ldots, \alpha_n)$ where:

1. B is a finite set of darts;
2. $\forall i, 0 \leq i \leq n, \alpha_i$ is an involution on B (an involution f on S is a mapping from S onto S such that $f = f^{-1}$);
3. $\forall i, j, 0 \leq i < i+2 \leq j \leq n, \alpha_i\alpha_j$ is an involution (condition of quasi-manifolds).

In Fig. 5, dart 1 corresponds to vertex s_1, edge a_1 and face f_1, dart $2 = \alpha_0(1)$ corresponds to (s_2, a_1, f_1), $3 = \alpha_1(2)$ corresponds to (s_2, a_2, f_1), and $20 = \alpha_2(3)$ corresponds to (s_2, a_2, f_2). The following table shows the different links of the first ten darts in the figure:

	1	2	3	4	5	6	7	8	9	10
α_0	2	1	4	3	6	5	8	7	10	9
α_1	5	3	2	8	1	7	6	4	18	13
α_2	1	2	20	19	5	6	7	8	9	10

For all the darts of the first line, we show in the following lines the dart to which it is α_0-, α_1- and α_2-linked with. We can verify that each α is an involution.

Let G be an n-g-map, and S be the corresponding subdivision. A dart of G corresponds to an $(n+1)$-tuple of cells $(c_0, \ldots, c_n)$, where c_i is an i-dimensional cell that belongs to the boundary of c_{i+1} [Bri89]. α_i associates darts corresponding with $(c_0, \ldots, c_n)$ and $(c'_0, \ldots, c'_n)$, where $c_j = c'_j$ for $j \neq i$, and $c_i \neq c'_i$ (α_i swaps the pair of i-cells that are adjacent to the same $(i-1)$ and $(i+1)$-cells). When two darts b_1 and b_2 verify $\alpha_i(b_1) = b_2$ $(0 \leq i \leq n)$, b_1 is said i-sewn with b_2. Moreover, if $b_1 = b_2$ (that is, b_1 is a fixed point) then b_1 is said i-free (closed objects do not exhibit any fixed points ; For open objects, the fixed points by α_n form the boundary of the object). n-g-maps provide an implicit representation of cells:

Definition (orbit) Let $\{\Pi_0, \ldots, \Pi_n\}$ be a set of permutations on B. The orbit of an element b relatively to this set of permutations is $\langle \Pi_0, \ldots, \Pi_n \rangle (b) = \{\Phi(b), \Phi \in \langle \Pi_0, \ldots, \Pi_n \rangle\}$, where $\langle \Pi_0, \ldots, \Pi_n \rangle$ denotes the group of permutations generated by $\{\Pi_0, \ldots, \Pi_n\}$.

Definition (i-cell) Let G be an n-g-map, b a dart and $i \in N = \{0, ., n\}$. The i-cell adjacent to b is the orbit $\langle \rangle_{N-\{i\}}(b) = \langle \alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n \rangle (b)$.

An i-cell is the set of all darts that can be reached starting from b, using any combination of all involutions except α_i. In Fig. 5, the 0-cell (vertex) adjacent to dart 2 is the orbit $\langle \alpha_1, \alpha_2 \rangle (2) = \{2, 3, 20, 21\}$, the 1-cell (edge) adjacent to dart 3 is $\langle \alpha_0, \alpha_2 \rangle (3) = \{3, 4, 19, 20\}$, and the 2-cell (face) adjacent to dart 9 is $\langle \alpha_0, \alpha_1 \rangle (9) = \{9, 10, 13, 14, 17, 18\}$. The set of i-cells is a partition of the darts of the n-g-map, for each i between 0 and n. Two cells are disjoined if their intersection is empty, i.e. when no dart is shared by these cells. More details about n-g-maps are provided in [Lie94].

2.2. Embeddings

The topological model only describes the structure of an object. The geometric information, such as the vertices positions for instance, must be added. For that purpose, it is possible to embed information in the structure, more precisely information is attached to darts. However, the information is usually related to a cell (in other words an orbit). This means that all the darts of an orbit share the same information. In practice, we can either attach the same information to all the darts of the orbit or to a single dart (we have used the second option, as discussed in the implementation section below). As a consequence, a dart can potentially embed information for each cell to which it belongs. This information can be geometric, but any other type of information is possible (chromaticity used for rendering for instance). In our context, mechanical embedding is appropriate.

3. Topology-based physical modeling

3.1. Discussion

Using topology-based models, the modeling process may only focus on mechanical embeddings since the complete structure (vertices, edges, faces and volumes) is represented within the model. Mechanical attributes must be added to each considered cell. For instance, the mass of an element is included within the volume embedding, the mass of a node in the vertex embedding, the stiffness of a spring in the edge embedding, and so on.

However, this approach requires that the mechanical structure corresponds to a quasi-manifold. As stated in the previous work section, the mesh of a finite-element model is recommended to obtain such a structure. Nevertheless, this hypothesis is more restrictive for particles models, since the interaction graph is not likely to be quasi-manifold [ES00].

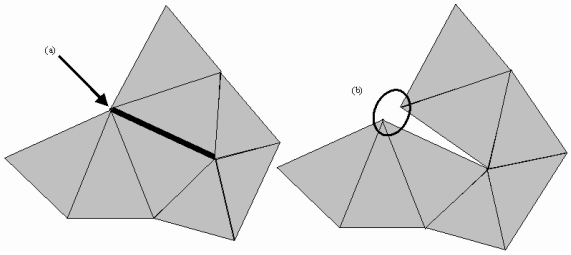

Figure 6: *While applying the cutting (a) on a 2D mesh, a vertex (b) can be dissociated in two.*

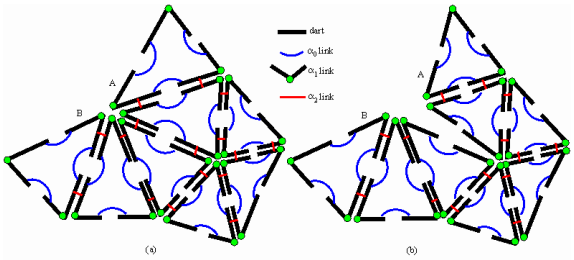

Figure 7: *In the 2D case (a), we separate two faces by cancelling the α_2 link. As can be seen in (b), the darts A et B no longer belong to the same vertex orbit.*

Basically, this means that nodes should be vertices, springs are mapped to edges, and faces and volumes are formed by a set of nodes and springs. For instance, in the model proposed by [Pro95], neither a spring that crosses a face corresponds to an edge nor a spring that links second neighbors (that is, neighbors of neighbors). By relying on topological models, we have to accept that only a sub-part of the mechanical structure should be in bijection with the topological structure. The remaining mechanical elements should be embedded in a way as convenient as possible. For instance, a spring that crosses a face should be embedded in the face; a spring which links second neighbors should be embedded on the common node/vertex or one of the two concerned edges, etc.

In the remainder of the paper, we suppose that the mechanical structure is quasi-manifold, that can be any Finite-Element mesh or a mass/spring system where any interaction maps to an edge.

3.2. Topological operations

In section 1, we have shown the need for topological operations in surgical simulation. Here, we show that these operations can be very efficiently implemented using *n*-g-maps.

Let us consider the volume separation (face dissociation) first. Using *n*-g-maps, this transformation only corresponds to the α_3-unsew operation between two volumes and is really straightforward: The algorithm walks through the darts of the face (exploring α_0 and α_1 links), and discards α_3 links.

During this process, the elimination of α_3 may induce vertex and/or edge dissociation, as shown in Fig. 6 in the 2D case. Fig. 7 shows what is happening in the generalized map when we eliminate adjacency relationships and a vertex is dissociated. This side-effect can be easily detected, since in this case, two unsewed darts no longer belong to the same vertex (and/or edge) orbit. This must be checked during the last step of the operation, since we must prevent unsewed darts to share the same embeddings. The following steps are performed:

- get a dart for each vertex and edge of the face to dissociate
- get a corresponding dart for each vertex and edge of the adjacent face, by following α_3 links
- for each dart d of the face and adjacent face, eliminate $\alpha_3(d)$
- check if a vertex is dissociated by checking if its dart and the corresponding dart on the adjacent face still belong to the same vertex orbit. If not, separate vertex embeddings.
- check if an edge is dissociated by checking if its dart and the corresponding dart on the adjacent face still belong to the same edge orbit. If not, separate edge embeddings.

When an orbit is dissociated, its embedding is available for only one of both resulting orbits. As far as geometric embeddings are concerned, another embedding must be created by duplication for the other connected part of the orbit. In practice, this basically means that the two resulting orbits are topologically disjoined but share the same geometry, in other words, the same space coordinates. As far as mechanical embeddings are concerned, the duplication of embedding is irrelevant. A new embedding of each part of the dissociated orbit must be computed according to the mechanical properties of the modified object. For instance, the mass of the original vertex must be distributed between the two resulting vertices. A similar process is necessary for stiffness in the case of edge dissociation.

The other operation we have considered is volume removal. A straightforward but naive approach consists in erasing all the darts of the volume. However, it can be quite hazardous to control embeddings correctly using such an approach. When a dart is deleted, all its embeddings are dispatched on other darts belonging to its initial *i*-cell. However, as seen previously, mechanical embeddings should be adapted accordingly. Unfortunately, that task becomes tricky because of the loss of information implied by the deletion of the darts. It is more convenient to isolate the volume first, by iterating on all its faces and applying the previously-defined face dissociation. Such a technique guarantees to share mass, stiffness and so on. When the volume is isolated, all its darts and corresponding embeddings can be deleted. Note that the unsewed faces of the volumes surrounding the removed vol-

ume become 3-free and must be taken into account in the rendering step (since they belong to the boundary of the object).

4. A topology-based mass/spring system

In this section, we show how to build a simple mechanical model, namely a mass/spring mesh, upon a topological model. We first have to consider how to embed the mechanical properties and state variables into the structure. At this point, we can study the basic algorithms to compute the behavior of the mesh. Then we present the topological operations that the model can currently undergo.

4.1. Embeddings

As explained in section 3, we choose to consider a mass/spring mesh as a quasi-manifold. Here, we suppose that the mesh is only composed of tetrahedra, but our model can easily be extended to other volume elements, since *n*-g-maps support any geometrical shape of quasi-manifolds.

We associate each mechanical property with the corresponding cell. In a tetrahedron, we embed its mass (which is easily obtained from the volume of the element for an homogeneous object). In an edge, we embed the rest-length of the corresponding spring, the stiffness and, if needed, a damping value. We select the outside faces of the object and embed into them some details required by the rendering step, such as faces normals for instance. Finally in a vertex, we embed the mass of the corresponding node (obtained by summing the contribution of all surrounding tetrahedra), an ambient viscosity coefficient and the position and velocity of the node. It is wise, for optimization purposes, to also include the sum of all forces applied to a node.

4.2. Behavior computation

Since different mechanical properties and state variables are embedded into the structure, all usual algorithms must be designed once again. Indeed, neither nodes nor springs appear as arrays that we only have to parse. On the contrary, each algorithm must rely on an adapted coverage of the topological structure.

A coverage of the topological structure is always defined from a starting dart and a given set of α links. To cover the whole *i*-cell, we basically check every dart of the structure, process the corresponding *i*-cell and mark all darts that belong to the same *i*-cell orbit (i.e. that can be reached by a combination of α_j , $\forall j \in \{0,...,n\} - \{i\}$). While walking through the structure, only unmarked darts are taken into account before they are marked. At the end, all darts are marked. At this stage, another coverage of the structure aims at unmarking the darts (or alternatively, the value of the marker is only swapped) and the marker can be released for further uses.

To compute deformation forces, we cover all the edges of the structure. Let b a dart that belongs to an edge orbit. We can easily get the extremity nodes of the corresponding spring: b belongs to the vertex orbit of one of the nodes, and $\alpha_0(b)$ to the other one. We can then compute the deformation force relying on the position and velocity of extremity nodes and apply it.

Force integration and state variables computation require to cover all vertices in a similar way. The process is rather straightforward for explicit integration schemes such as Euler or Runge Kutta. However, more elaborate integration methods rely on linear algebra. For such methods, it is recommended to associate each vertex with an index in the state variables vector. A final walk through the structure allows each vertex to retrieve its new position and velocity in the state vector.

Finally, because the collision process can be handled in different ways [TKH*05], we cannot make a complete inventory of all possible approaches. Depending on the chosen one, cells of a given dimension must be walked through. For instance, one can cover vertices and faces to check if a collision occur between objects. In penalty-based method, it is more appropriate to use a volumetric approach, so the volumes must be covered. Anyway, the topological model always allows us to find the concerned vertices, get their position and velocity to compute the collision correctly and apply resulting forces.

4.3. Topological operations

We have shown in section 3 that the topological operations we want to implement are topologically straightforward but require a careful handling of embeddings during face dissociation.

We present in this section how to find the new embeddings when a face is dissociated to separate two tetrahedra. Each triangle of a tetrahedron is composed of 6 darts (2 darts per edge). These 6 darts are separated (α_3-unsewed) from the 6 darts of the common triangle with the adjacent tetrahedron. For each vertex and each edge of the triangle, we check if the corresponding unlinked darts still belong to the same orbit. If it is not the case, a vertex or edge dissociation has occurred (see Fig. 8).

When a vertex is dissociated, we duplicate its position and velocity but we have to compute the mass of the two new vertices. Mass is obtained by covering adjacent tetrahedra and sum up their respective mass contribution.

$$m^v = \sum_{i \in tetra(v)} m_i^t / 4 \qquad (1)$$

where m^v is the mass of a vertex v, $tetra(v)$ the tetrahedra adjacent to v and m^t the mass of a tetrahedron. For the sake of

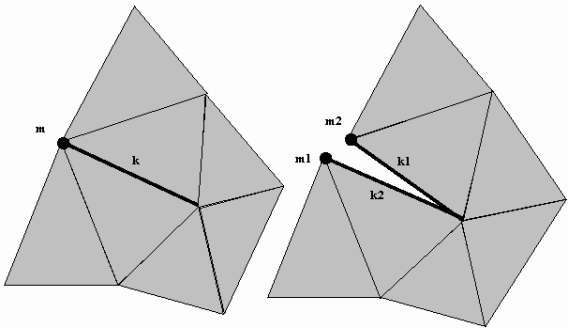

Figure 8: *After the dissociation of a vertex and an edge, new masses m and stiffnesses k must be computed. Here, we have $m = m_1 + m_2$ and $k = k_1 + k_2$.*

simplicity, we found convenient to dispatch 1/4 of the mass of a tetrahedron to each of its vertices. A more complex approach could have been used: We could have considered the part of volume related to each vertex (by splitting the tetrahedron according to the mediator planes of each edge) and distribute the mass accordingly. Note that this computation must be done considering the rest shape of the object and not after deformation (the model does not simulate mass transfer). This approach guarantees a better mass repartition in case of thin parts of a tetrahedron. However, in physically-based modeling, these kind of shapes are known to lead to ill-conditioned equations and must be avoided, so we can suppose that a heterogeneous distribution of mass is a reasonable hypothesis.

This way, we ensure that the sum of the masses of the two vertices is always the mass of the original dissociated vertex. We then consider that ambient viscosity is proportional to the computed mass. Note that if the vertices have an index into a state vector, we have to supply one of the appearing vertices with a new index (the other vertex keeps the index of the dissociated vertex).

The edge dissociation process is rather similar. If two unlinked darts no longer belong to the same edge orbit, an edge dissociation has occurred. We then consider all tetrahedra adjacent to both resulting edges and sum their contributions to obtain the resulting stiffness. This way, the sum of the stiffness of the two edges equals the stiffness of the initial dissociated edge, as stated by the Kirchhoff law applied to a parallel combination of springs.

5. Implementation and optimizations

5.1. Implementation

We have implemented the model described in the previous section. To handle 3-g-maps, we used the MOKA library (freely available at *http://moka-modeller.sourceforge.net*). The mechanical aspects and the interaction tools are handled within a home-made dynamic simulator [MDH*03] , using penalty methods between tetrahedra to handle collisions. The Fig. 9 shows our results for a liver object after several tetrahedron removals.

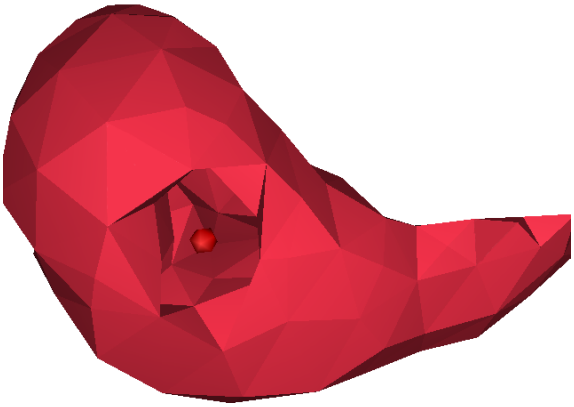

Figure 9: *Example of the removal of several tetrahedra.*

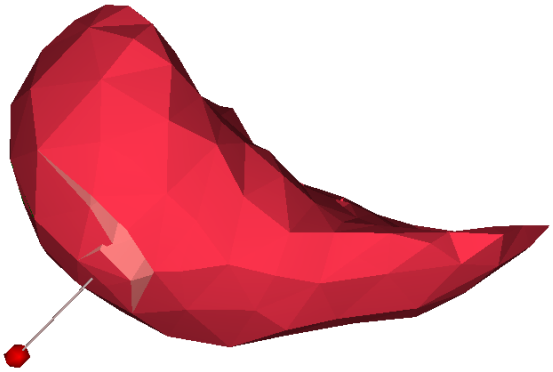

Figure 10: *Example of several dissociations of faces around a vertex.*

Fig. 10 shows an example of a tearing. A simple face dissociation to separate two volumes does not provide any convincing visual effect, especially when no vertex is dissociated. To allow the volumes to really separate from each other, we have to dissociate several adjacent faces. The idea is to extract a closed triangle fan around a vertex where the inner forces go beyond a given threshold. Dissociating all the faces of the fan yields this vertex to be dissociated. This fan is easily defined using our topology model. Indeed, we find a vertex we want to dissociate and select a starting face by choosing, among adjacent faces, the one whose normal best approximates the inner stress direction. Let d_0 be one of the two darts belonging to both face and vertex orbits. We find a dart d_1 of the next face in the fan by applying α_2, α_3

and again α_2 and α_1 successively to d_0. We iterate to get the next d_i in order to find the other faces of the fan. The process stops when the initial dart d_0 is reached or when the surface of the object is met. In this last case, the process must stop before applying α_3. We then apply the same algorithm to $\alpha_1(d_0)$ to get the other part of the fan. Since no α_0 is used, we ensure that we obtain darts belonging to the orbit of the vertex to dissociated.

First, we have implemented all algorithms with the abovementioned coverages of the structure. Using a dual core 2.8GHz processor, we needed 12ms to compute an evolution step of a deformable body including 564 nodes, 2850 springs and 1856 tetrahedra (using Runge Kutta 4). This computation time must be compared with the 2ms that the same object based on a classic adjacency graph requires. This slowdown is explained by the number of darts (here 44544) that must be checked during the coverage of any type of cell. Optimizations are obviously needed to avoid such coverages, even if the faster model lacks a lot of functionalities considering the poorness of the adjacency relationships it brings (we have not ventured to implement topology changes on this naive model).

5.2. Optimizations

To get faster results, we have chosen to put all the embeddings of cells of a given dimension in a separate double-linked list (easy to implement using construction patterns without disturbing topological operations). This prevents us from covering all the structure where only embeddings (and not the darts containing them) are needed. This optimization allows us to reach 8ms for a simulation step, but this is clearly not sufficient. A lot of algorithm steps require not only embeddings but also neighborhood relations that embeddings are not intended to provide, so this optimization appears quite limited.

Considering processor architectures and memory caching, it is often recommended to put data into contiguous and indexed structures such as arrays. It is not worth trying to make our map-based model (a graph structure) perform as fast as an indexed adjacency graph. In other words, the simulation loop should preferably rely on arrays while the topology modifications should rather be driven by the topological model. Here, we discuss how this idea can be implemented in practice.

A first approach consists in manipulating both models altogether. This implies that any topological operation applied to the topological model should also be applied to the indexed structure. This method is not really relevant. Indeed, since the basic element of the topological model, namely the dart, does not correspond to anything in the indexed model, it is not possible to translate the atomic topological operations (α_i-sewing or unsewing) into modifications in the adjacency graph. This method forces the topological changes to be expressed in terms of cell modifications, which is definitely not intuitive when using (g-)maps and really restricting. This leads to define new non-robust algorithms to modify the object.

A second approach still consists in driving the topological modification using the topological model, which guarantees robustness and consistency. Nevertheless, when all the modifications have been applied, we project the map into an indexed structure which is not intended to duplicate the information but rather aims at providing a fast way to navigate inside the 3-g-map and access data. We therefore define four arrays (one for each dimension of cells: vertices, edges, faces and volumes). In these arrays, we find a pointer to a dart representing each *i*-cell and another pointer to the corresponding embedding. The proposed data structure is defined as follows:

```
struct Cell {
  Dart *dart;
  Embedding *emb;
};
vector<Cell> vertices,edges,faces,volumes;
```

This indexed structure is built from scratch after a series of simultaneous topological changes and is used as such for following simulation steps until a new topological modification occurs. The construction phase requires a little amount of time which is negligible compared to the overall computation time required to apply topological changes.

Using this optimized structure, we have been able to almost reach the results obtained with indexed structure (2ms), the possible overhead (due to the use of indirections such as pointers) is below the measure noise (10%). Thus, our optimization method offers a satisfactory trade-off between robustness and computation time.

Conclusion and future work

In this paper, we have presented a framework based on combinatorial topology that mechanical models can rely on. Since maps represent any adjacency relationship explicitly, they provide a convenient way to address topological issues of models with variable topology. The computation of neighborhood properties is simplified as well as the design of operations. We have shown how to build the mechanical model on this framework, by embedding physical properties and state variables in the structure and have designed our algorithms in terms of coverages of the structure. We have shown how to compute the new embeddings when some topological change is applied.

Using adequate optimization, we find that our model is as fast as a naive indexed approach based on adjacency graphs. In this paper, we only showed basic modifications, but more complex ones can be considered. For instance, the loss of connectivity is quite simple to verify: It consists in checking if all the darts are marked while walking through the structure following all the α-links. When the connectivity is

lost, the mechanical model can be split in two parts to compute the behavior of the two resulting bodies separately (and even computing collisions between these two components). Another example of useful complex topological operation is refinement, that offers a good trade-off between precision and shapes of elements as showed by [FDA05].

Moreover, we now aim at implementing an explicit linear finite-element model upon our topological basis (such as the co-rotational approach [MG04]). Whereas mechanical embeddings of such a model seem straightforward (position and velocity on vertices, mass and local matrices on tetrahedra), the inherent requirement of linear algebra during the resolution process implies to provide each vertex with an index, and moreover, to number vertices efficiently. Since this process does not fit the topological structure well, we hope to keep the resolution efficient even if "on-the-fly" numbering is used.

Acknowledgment

This work has been funded by the French Research Agency through the VORTISS Project (ANR-06-MDCA-015). We also want to thank the IRCAD (Strasbourg) for the liver model and G. Damiand for helping us to use MOKA.

References

[Bau75] BAUMGART B. G.: A polyhedron representation for computer vision. In *AFIPS'75 national computer conference and exposition* (May 1975), pp. 589–596. 3

[BG00] BIELSER D., GROSS M. H.: Interactive simulation of surgical cuts. In *Pacific Graphics 2000* (2000), pp. 116–125. 2

[Bri89] BRISSON E.: Representing geometric structures in d dimensions: topology and order. In *Annual symposium on Computational geometry* (1989), pp. 218–227. 4

[CDA00] COTIN S., DELINGETTE H., AYACHE N.: A hybrid elastic model allowing real-time cutting, deformation and force-feedback for surgery training and simulation. *The Visual Computer 16*, 8 (2000), 437–452. 2

[DSCP94] DELINGETTE H., SUBSOL G., COTIN S., PIGNON J.: A craniofacial surgery simulation testbed. In *Visualization and Biomedical Computing* (Oct. 1994), pp. 607–18. 3

[Edm60] EDMONDS J.: A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society 7* (1960). 3

[ES00] EGLI R., STEWART N. F.: A framework for system specification using chains on cell complexes. *Computer-Aided Design 32* (2000), 447–459. 3, 4

[FDA05] FOREST C., DELINGETTE H., AYACHE N.: Removing tetrahedra from manifold tetrahedralisation: application to real-time surgical simulation. *Medical Image Analysis 9*, 2 (Apr. 2005), 113–122. 2, 9

[FG99] FRISKEN-GIBSON S.: Using linked volumes to model object collisions, deformation, cutting, carving and joining. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (Oct. 1999), 333–348. 2

[JCD09] JUND T., CAZIER D., DUFOURD J.-F.: Particle-based forecast mechanism for continuous collision detection in deformable environments. In *Geometric and Physical Modeling* (Oct. 2009). 1

[LFB08] LIENHARDT P., FUCHS L., BERTRAND Y.: Combinatorial models for topology-based geometric modeling. In *Theory and Applications of Proximity, Nearness and Uniformity*, vol. 23. Dipartimento di Matematica, Seconda Universita di Napoli, Dec. 2008. 2

[Lie91] LIENHARDT P.: Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer Aided Design 23* (1991), 59–82. 3

[Lie94] LIENHARDT P.: N-dimensional generalized combinatorial maps and cellular quasimanifolds. *Int. J. of Computational Geometry and Applications 4*, 3 (1994), 275–324. 3, 4

[LSM08] LÉON P.-F., SKAPIN X., MESEURE P.: A topology-based animation model for the description of 2d models with a dynamic structure. In *Workshop on VR Interactions and Physical Simulation* (Nov. 2008), pp. 67–76. 1

[MBF04] MOLINO N., BAO Z., FEDKIW R.: A virtual node algorithm for changing mesh topology during simulation. *ACM Transaction on Graphics (SIGGRAPH) 23*, 3 (2004), 385–392. 2

[MDH*03] MESEURE P., DAVANNE J., HILDE L., LENOIR J., FRANCE L., TRIQUET F., CHAILLOU C.: A physically-based virtual environment dedicated to surgical simulation. In *Int. Symp. on Surgery Simulation and Soft Tissue Modeling* (June 2003), vol. 2673 of *Lecture Notes in Computer Science*, pp. 38–47. 7

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Graphics Interface'04* (2004), pp. 239–246. 9

[NTB*91] NORTON A., TURK G., BACON B., GERTH J., SWEENEY P.: Animation of fracture by physical modeling. *The Visual Computer 7* (1991), 210–219. 2

[NvdS00] NIENHUYS H., VAN DER STAPPEN A.: Combining finite element deformation with cutting for surgery simulations. In *EUROGRAPHICS'00 (short presentations)* (Aug. 2000), pp. 43–51. 2

[NvdS03] NIENHUYS H., VAN DER STAPPEN A.: A delaunay approach to interactive cutting in triangulated surfaces. In *Int. Workshop on Algorithmic Foundations of Robotics* (2003), vol. 7, Springer Verlag, pp. 113–130. 2

[OH99] O'BRIEN J., HODGINS J.: Graphical modeling and animation of brittle fracture. In *SIGGRAPH 99 Conference Proceedings* (Aug. 1999), Computer Graphics annual conference series, pp. 137–146. 2

[Pro95] PROVOT X.: Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics Interface'95* (May 1995), pp. 147–154. 5

[SDF07] SIFAKIS E., DER K., FEDKIW R.: Arbitrary cutting of deformable tetrahedralized objects. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), pp. 73–80. 2

[SOG06] STEINEMANN D., OTADUY M. A., GROSS M.: Fast arbitrary splitting of deforming objects. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006), pp. 63–72. 2

[TF88] TERZOPOULOS D., FLEISHER K.: Modeling inelastic deformation : Viscoelasticity, plasticity, fracture. *Computer Graphics (Proceedings of ACM SIGGRAPH 88) 22*, 4 (Aug. 1988), 269–278. 2

[TKH*05] TESCHNER M., KIMMERLE S., HEIDELBERGER B.,

ZACHMANN G., LAKS L. R., FUHRMAN A., CANI M.-P., FAURE F., MAGNENAT-THALMANN N., STRASSER W., VOLINO P.: Collision detection for deformable objects. *Computer Graphics Forum 24*, 1 (2005), 61–81. 6

[Ton91] TONNESEN D.: Modeling liquids and solids using thermal particles. In *Graphics Interface'91* (June 1991), pp. 255–262. 2

[TPF89] TERZOPOULOS D., PLATT J., FLEISHER K.: Heating and melting deformable models (from goop to glop). In *Graphics Interface'89* (1989), pp. 219–226. 2

Appendix A: C++ Code of a topological operation

We present here the example of a separation of volumes by dissociating their common face in order to illustrate that the implementation of topological operation is simple and straightforward. This example heavily relies on MOKA API.

```
void dissociateFace(Dart* a_dart) {
  // a_dart is any of the darts of the face
  // to dissociate (from its adjacent face)

  int n=a_dart->NumberOfFaceVertices();
    // number of vertices and edges
    // of the face
  Dart* da[n];
    // each dart represents
    // a vertex and an edge of the face
  Dart* db[n];
    // each dart represents a
    // vertex and an edge of
    // the adjacent face

  // get one dart for each
  // vertex and edge of the face
  // (by covering alpha0 and alpha1 links)
  da[0]=a_dart;
  for(int i=1;i<n;i++)
    da[i]=da[i-1]->getAlpha0()->getAlpha1();

  // get the corresponding darts
  // on the adjacent face
  // following alpha3 links
  for(int i=0;i<n;i++)
    db[i]=da[i]->getAlpha3();

  // **** Separate the two volumes ****
  for(int i=0;i<n;i++) {
    da[i]->setAlpha3(NULL);
      // means free alpha3 link
    db[i]->setAlpha3(NULL);

    // Each edge has two darts
    // (linked by alpha0) on the face.
    // alpha3 must also be changed
    // for these darts to maintain
    // the 3-g-map consistency
    da[i]->getAlpha0()->setAlpha3(NULL);
    db[i]->getAlpha0()->setAlpha3(NULL);
  }
  // separation is complete

  // Check vertex dissociation
  for(int i=0;i<n;i++) {
    if (!da[i]->isSameOrbit(VERTEX_ORBIT,db[i])
      // separate vertex embeddings
  }

  // Check edge dissociation
  for(int i=0;i<n;i++) {
    if (!da[i]->isSameOrbit(EDGE_ORBIT,db[i])
      // separate edge embeddings
  }
}
```

In this algorithm, the *isSameOrbit* method consists in walking along the orbit given as the first parameter of the function and checking if the dart provided as the second parameter is met.