Adaptive Resolution for Topology Modifications in Physically-based Animation

Philippe Meseure, Emmanuelle Darles, Xavier Skapin, Yazid Touileb

Adaptive Resolution for Topology Modifications in Physically-based Animation

P. Meseure[1], E. Darles[1], X. Skapin[1], Y. Touileb[2]
[1]University of Poitiers, XLIM CNRS Lab., SIC department, France
[2]University of Limoges, XLIM CNRS Lab., SIC department, France

XLIM Technical Report

June 29, 2015

This article has been successively submitted to *Computer Graphics Forum* (response received on May, 31st 2014) and *The Visual Computer* (response received on August, 27th 2014) without success. Meanwhile, Fléchon et al. published similar work at Vriphys, in September 2014 [FZDJ14]. Since there are only slight conceptual differences between their work and ours, but different algorithmic approaches and hypotheses on the model, we have decided to make our paper freely available on the web. In a section added at the end of the paper, the reader will find more information that helps positioning it among recent work.
Please contact `Philippe.Meseure@univ-poitiers.fr` for further details.

Abstract

This paper shows the interest of basing a mechanical mesh upon an efficient topological model in order to give any simulation the ability to refine this mesh locally and apply topological modifications such as cutting, tear and matter destruction. Refinement and modifications can indeed be combined in order to get a more precise result. The powerful combinatorial map model provides the mathematical background which ensures that the quasi-manifold property is guaranteed for the mesh after any topological modification. The obtained results offer the versatility and time efficiency that are expected in applications such as surgical simulation.

keywords: Physically-based animation, Adaptive/hierarchical models, Generalised maps, Topological modifications, surgical simulation.

Introduction

Much research has been focused on physically-based deformable models for interactive manipulation for twenty years. Nowadays, simulators require models that can not only be computed in real time, but also undergo structure modifications. Several methods have been proposed to compute the behaviour of deformable bodies, such as Finite Element Method (FEM), Finite Differences [DDCB01], mass/spring systems [Pro95], and so on. Some of these models allow structure modifications such as fractures [TF88, NTB$^+$91]. This is useful in different applications but more specifically in surgical simulation, where cuttings, tears or matter destruction are undergone by physical bodies.

Several topological modifications have been proposed, as shown in Figure 1. Basically, face splitting or volume deletion have been used. But these approaches suffer from the coarse resolution of the simulated mechanical mesh (needed to ensure interactive manipulations). The cutting of some volumes of the mesh by a plane surely gives an acceptable visualisation of the cut path [BG00, GCMS00, MK00, SOG06, BGTG03], but heavily disturbs the mechanical simulation by creating non regular or ill-formed elements.

Besides, some approaches have been proposed to refine this mesh [HPH96, FDA02] or provide several levels of detail of simulation [DDCB01]. The main goal of these models is to adapt the mesh resolution, depending on the focus and the modality of interactions. While a fine definition of geometry is locally required in any contact zone, other areas can be computed at a coarser level. This adaptation allows to devote most of calculations to areas of interest, and therefore saves computation time.

Unfortunately, very few approaches allow topology changes in refined meshes [FDA05, GCMS00]. Dick *et al.* rely on a hexahedral mesh refined by means of an octree structure. General models with adaptive resolution require a complex structure that makes modifications quite hazardous. Defining robust and stable topological changes in such structures appears to be a real challenge.

In this paper, we propose to base any meshed mechanical system on a topological model (namely, generalised maps), to achieve real-time interaction and robust modifications of a structure with adaptive resolution. Our contributions include:

1. a topological model with adaptive resolution and the associated mechanical embedding, allowing various interactive simulations of deformable bodies (either discrete or continuous). The model is mathematically-defined, which guarantees both its self-consistency and robustness,

2. a refinement process that allows the adaptation of the model during runtime,

3. several different topological operations that allow interactive structure modifications while still guaranteeing consistency.

The article is organised as follows. First, we discuss the work dealing with physical simulation including, on the one hand, topological modifications, and,

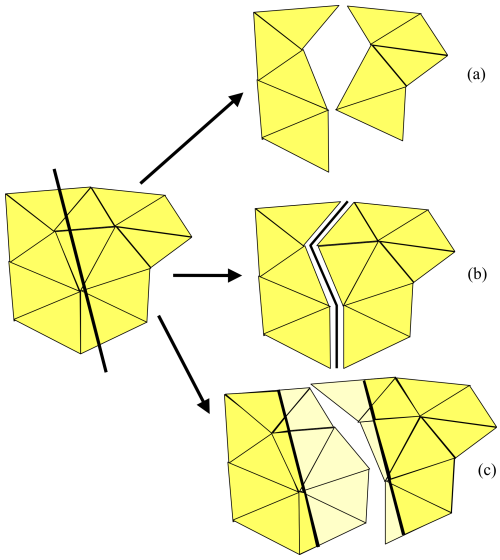

Figure 1: Different topological approaches for cutting in a $2D$ mesh : from top to bottom, (a) face deletion, (b) face separation (edge splitting) and (c) face duplication

on the other hand, adaptive or multiresolution approaches. In a second section, the main principles to provide a physical simulation with a topological basis are explained. In section 3, the topological model allowing adaptive resolution is described. In the following section, the refinement process is detailed, including the associated mechanical adaptation. Next, in section 5, the operations for cutting/tearing and matter destruction are presented. Finally, some results and performance measures are shown before conclusion and prospects.

1 Previous work

1.1 Physical simulation including topology changes

Various physical models have been proposed to simulate objects deformation in Computer Graphics. Two types of methods are often compared. First, so-called discrete methods rely on a set of point masses and interactions between all or a subset of couples of these points (for instance, see [LJF+91] as a general approach). Among them, mass/spring systems (MSS) rely on a mesh where vertices are provided with mass and edges and/or faces with damped springs. Second, Finite Differences or Finite Element Methods (FEM) can be used to solve the equations of continuous mechanics [TPBF87, OBH02]. These models have gained interest for interactive simulation since the introduction of explicit FEM [CDA00], that compute forces resulting from the deformation of volumetric

elements and apply them to the vertices of the mesh. Co-rotational computation [MG04] and several consecutive approaches [NPF05] provide more realistic behaviours by determining a rigid displacement of elements before computing their deformation. From an algorithmic point of view, discrete and continuous simulations of a deformable mesh only differ by the way the forces applied to nodes are computed, but the overall simulation algorithm is the same.

As mentioned in [SHGS06], there are three main approaches to simulate cutting. The first category consists in deleting cells intersecting a cut path [MK00, CDA00]. These methods are well-suited for matter destruction but hardly represent a sharp cut, since they loose matter and, as a consequence, mass. The second category of approaches separate volumes [FG99, NvdS00]. These methods generate a dubious cut trace, approximated by the local geometry of the mesh. They have been enhanced by "node snapping" that consists in deforming the initial mesh to match a cut path [SHS01, NvdS03]. Some approaches directly cut the mesh elements [BG00, GCMS00, MK00, BGTG03]. A mesh co-refinement can also be used [SOG06]. However, all these methods create sliver elements that the force model (either discrete or continuous) is unable to correctly compute [Bat82].

To avoid such ill-conditioned elements, some authors [FDA02, DGW11] propose to refine the mesh in the cut area. This approach guarantees to keep elements with right proportions while allowing a good approximation of the cut path. The "virtual node algorithm" is another way to avoid ill-conditioned elements. It was first introduced by [MBF04] and extended by [TSSB$^+$05, SDF07]. It consists in duplicating elements intersected by a cut plane and adapting their mechanical behaviour depending on which side of the plane they are placed. It does not modify the element shapes but the duplication results in a non-manifold mesh (neighbour volumes can be adjacent to both copies of the initial volume by the same face). Since manifold property cannot be assumed, it is hard to check if the topology of the mesh is consistent after a modification.

A recent survey of cuts simulation can be found in [WWD14]. All the above-mentioned methods rely on a simple topological model. Indeed, to compute the model's forces, adjacency relations are required. An explicit encoding of cells (vertices, edges, faces, volumes) and adjacency graphs are often used. Each cell is identified by an index pointing to a table giving the index of its adjacent cells (volumes pointing to vertices for FEM or to edges for MSS). Unfortunately, such a simple structure allows the modelling of non-manifold objects, which makes topological modifications a tedious process. For instance, Forest *et al.* use a heavy post-processing to ensure the manifold property of their mesh [FDA05]. A more robust topological model, that is, a model that can only represent manifolds (or, more exactly, quasi-manifolds) and nothing else, is obviously desirable. Some recent works have proposed to rely on *combinatorial maps* [FZDJ13] (as an extension of half-edges) or *generalised maps* [MDS10] but, still, only simple modifications of the mesh (volume separation or deletion) have been proposed.

1.2 Refinement and adaptive resolution

As stated above, it seems that using subdivision or a finer resolution of a mesh is a convenient way to deal with both precision and well-conditioned elements. Note that using finer elements requires to integrate equations using a smaller time step (since mass is smaller and stiffness is higher). In [DDCB01], Debunne *et al.* propose to define a body at different resolutions and choose the most appropriate one, depending on the required precision in a given area. Interpolation is used to compute the effect of a deformation on the other resolutions. This model is unfortunately incompatible with topology modifications since the different resolutions of the mesh are computed separately and are not correlated one to another.

Some approaches rather rely on pyramidal models, that is, models where any finer resolution is always included in the coarsest ones [KCB09]. When the mesh is conformal, any introduction of a new vertex on a edge or a face implies to subdivide the surrounding volumes to take that vertex into account. Forest *et al.* [FDA02] use such a condition and give several subdivision schemes to match the subdivision applied to adjacent volumes. The other possibility is to define "inactive nodes" [HPH96, DMG05] between different resolutions. The position of such nodes is constrained, since they belong to non-subdivided elements. In other words, if, for instance, a vertex is located in the middle of an edge of a triangle, it should be kept there if the mesh elements are supposed to be triangles, whereas, topologically speaking, the element is no longer a triangle but a quad. In [DGW11], Dick *et al.* use a dedicated mechanical model based on FEM and given shape functions to handle the refined structure, but allow only one resolution level between to adjacent elements.

This article does not aim at presenting new modifications of mechanical meshes. Instead, it presents how generalised maps can efficiently endorse in a unique formalism various manipulations, such as adaptive resolutions and topology modifications, whatever the kind of the used mechanical laws (MSS or explicit FEM). The used meshes are supposed tetrahedral. To keep tetrahedra as well-proportioned as possible, we rely on an extension of the Loop subdivision scheme [Loo87], by splitting edges and building tetrahedra inside the original tetrahedron. The same scheme has been used in several methods, for instance [FDA02].

2 Topologically-based mechanical models

This section presents the concepts needed to understand the use of topological models in simulation. For further details, reader is invited to consult [MDS10] and [FZDJ13].

2.1 Generalised maps

A generalised map aims at representing quasi-manifolds, that is, for instance in $3D$, objects that are composed of volumes linked by (and only by) their faces

and where a face can link at most two volumes. These objects are subdivisions of space that can be decomposed into vertices, edges, faces and volumes. For convenience, the rest of the article refers to *i-cells* or cells of dimension i (0-cell means vertex, 1-cell edge, *etc.*). Figure 2 shows an example in $2D$. Note that $\alpha_0 \circ \alpha_2$ and $\alpha_2 \circ \alpha_0$ are the same permutation.

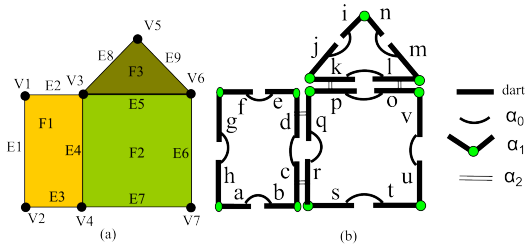

Figure 2: Example of a generalised map in $2D$

A generalised map or, more shortly, g-map is based on a unique abstract element, called *dart*. For $3D$-objects, a dart roughly represents "a vertex of an edge for a face and a volume of an object". In other words, a dart identifies a vertex, an edge, a face and a volume at the same time. There are as many darts as different quadruplets (vertex, edge, face, volume).

More formally, a g-map is defined in any dimension as a set of darts and permutations defined on these darts:

Definition 1 (Generalised map)**.** Let $n \geq 0$. A n-dimensional generalised map (or n-g-map) is defined as $G = (D, \alpha_0, \ldots, \alpha_n)$, where:

1. D is a finite set of darts,
2. $\forall i, 0 \leq i \leq n, \alpha_i$ is an involution,
3. $\forall i, j, 0 \leq i < i+2 \leq j \leq n, \alpha_i \circ \alpha_j$ is an involution (condition of quasi-manifolds).

Let G be an n-g-map. A dart of G corresponds to an $(n+1)$-tuple of cells $(c_0, ..., c_n)$, 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 $c_i \neq c'_i$. In other words, α_i allows one to point at a different i-cell while still pointing at the same cells of other dimensions.

When two darts d_1 and d_2 are such that $\alpha_i(d_1) = d_2$ $(0 \leq i \leq n)$, d_1 is said i-sewn with d_2. Moreover, if $d_1 = d_2$ then d_1 is said i-free.

α involutions can also be seen in a constructive way. α_0 is used to sew two darts (representing vertices) and create an edge. α_1 aims at connecting two edges by their common vertex. By repeating this operation, several edges can be 1-sewn in order to create a face when closing the obtained curve. α_2 aims at connecting faces by their common edge. Both darts of each edge are 2-sewn. The resulting edge includes four darts (for instance, edge $E4$ with darts c, d, q

and r in Figure 2). By connecting several faces by their edges and closing the obtained surface, a volume can be created. α_3 aims at sewing volumes together to form a complex $3D$ object.

A g-map provides an implicit representation of cells. Indeed, several darts point at the same i-cell. This set of darts is called *orbit* of this cell. For instance, a vertex is an orbit containing all the darts that are sewn by any α_1, α_2 and α_3 combination in $3D$. More formally, an i-cell associated with some dart d, is the set of all darts that can be reached starting from d, using any combination of all involutions except α_i. The set of i-cells, $0 \leq i \leq n$, is a partition of the g-map's darts. The notion of orbit can be extended to any combination of α_i to represent more than just i-cells. The list of α_i which are used to define an orbit is enclosed inside $<$ and $>$ brackets. For instance, given a dart d of a 3-g-map, $< \alpha_1, \alpha_2 > (d)$ contains darts pointing at different edges and faces, but at the same vertex and volume. It is therefore the orbit composed of all the faces surrounding a vertex, on a given volume. Note that $< \alpha_0, \alpha_1, \alpha_2, \alpha_3 > (d)$ represents the connected component of the object including dart d. More details about g-maps are provided in [Lie94].

A topological model only describes the structure of an object. Geometric information, such as a position for each vertex, must be added. For that purpose, it is possible to embed information in the g-map, that is, to attach information to all or a subset of its darts. However, the embedded information is usually related to a cell, in other words, an orbit. That means that the darts of an orbit share the same information. This information can be attached to all the darts of an orbit (*i.e.* all darts of this orbit include a pointer to the same information) or only to a single (randomised) dart of the orbit. However, this last approach requires to search, inside an orbit, for the dart which owns some requested information, which can be time-consuming. Note that any type of information can be embedded, not only geometric positions, but also colorimetric or rendering as well as mechanical information.

2.2 Mechanical Embedding

Mechanical information can also be embedded in the g-map. The used mechanical model is supposed to be a mesh composed of tetrahedra in $3D$ or triangles in $2D$. Depending on its dimension, the model relies either on a 3-g-map or a 2-g-map. All the vertices of the mesh support mechanical nodes. The fundamental equation of motion for each node i is:

$$m_i \mathbf{a}_i = \sum \mathbf{f}_{internal} + \sum \mathbf{f}_{external} \tag{1}$$

where m_i is the mass of the node, $\mathbf{a}_i$ the acceleration. The term $\sum \mathbf{f}_{external}$ includes ambient viscosity, gravity, *etc.* $\sum \mathbf{f}_{internal}$ represents internal deformation forces.

For each vertex, several data are embedded in its 0-cell: mass, ambient viscosity coefficient, rest position as well as runtime data (position, velocity, sum of all the applied forces, and so on). The mass of a node results from

the sum of the contributions of all its surrounding volumes in $3D$ or faces in $2D$ [GCMS00]. In $3D$, any tetrahedron is supposed to bring $1/4$ of its mass to each of its vertices. The mass of a tetrahedron depends on its volume, by considering that the volumetric mass is uniform. The same applies in $2D$, where triangles bring $1/3$ of their mass each of their vertices and the surface mass is supposed uniform.

Depending on the chosen deformation model, information can be embedded in cells of higher dimension. For instance, stiffness and damping can be included in some edge orbit if edges are supposed to correspond to damped springs. Explicit finite element method computes forces for each volume or face. Here, as a proof of concept, we consider linear elasticity for small displacements of an isotropic and homogeneous material represented by a tetrahedral mesh. Under these assumptions, for each tetrahedron, forces can be computed as:

$$\mathbf{f} = -\mathbf{K}\mathbf{u} \text{ with } \mathbf{K} = V\mathbf{B}^t\mathbf{C}\mathbf{B} \tag{2}$$

where V is the volume of the given tetrahedron at rest, $\mathbf{f}$ includes the forces for each node of this tetrahedron $(f_1^x,\ f_1^y,\ f_1^z,\ f_2^x,\ f_2^y,\ ...)^t$, and $\mathbf{u}$ represents the displacement from their rest position $(u_1^x,\ u_1^y,\ u_1^z,\ u_2^x,\ u_2^y,\ ...)^t$. $\mathbf{C}$ is the 6x6 stress-strain matrix that depends on the Young modulus and the Poisson coefficient. This matrix is the same for every tetrahedron. $\mathbf{B}$ is the 12x6 strain-displacement matrix allowing a measure of a deformation that is computed once and for all using the rest position of the vertices of each tetrahedron (see for instance [MG04] for more details about how to compute such matrices). $\mathbf{K}$ is the 12x12 rigidity matrix and has to be computed (as well as $\mathbf{B}$) for each tetrahedron and embedded in the corresponding 3-cell of the 3-g-map.

Simulation mainly consists in choosing the right path through the structure at each step. To compute internal forces, depending on the deformation model, a walk through the edges and/or volumes of the structure is needed. To compute collision, faces and/or volumes are covered, depending on the chosen collision detection. Integration phase (including velocity and position updates) requires to treat each vertex. One important point is that such a topological model is basically a graph where walking through the structure requires a costly marking process. To avoid such a waste of time, it is recommended to project the structure into arrays that allow an indexed cover of vertices, edges, faces and volumes (arrays are an order of magnitude faster to walk through than graphs). Note that indices in these arrays are only iterators and are not aimed at referring to cells. This structure has to be updated whenever (and only when) a topology change occurs. A fast way to handle such structures is to use arrays of embeddings [FZDJ13].

3 Adaptive Resolution Model

In this section, a new adaptive model that allows hierarchical subdivisions in a robust and consistent way is introduced. The structure being used is somehow related to Kraemer's hierarchical model [KCB09]. This model is dedicated

to multiresolution, and provides different permutations (in our case α_i involutions) depending on the chosen resolution. Moreover, a set of darts in a coarse resolution is always included in every set of darts of finer resolution (in other words, the model is nested). In practice, we use a simpler model since we do not use multiresolution, but the idea of defining resolution-dependent permutations provides us with a convenient way to control the refinement process.

We have chosen a Loop subdivision scheme [Loo87], extended to tetrahedra. First, it consists in applying it to each face of a tetrahedron. Then, one apex tetrahedron is built for each vertex of the original tetrahedron. These tetrahedra are similar to the original tetrahedron scaled uniformly by 1/2, so have strictly the same proportion. This guarantees that the obtained tetrahedra are well-conditioned for physical simulation. This refinement scheme also avoids providing vertices with new edges, that is, providing nodes with new links (springs), which generally results in system instabilities. After building apex tetrahedra, an octahedron appears in the middle of the tetrahedron [CMQ02]. This octahedron is thereafter subdivided into four tetrahedra by means of a process described in section 4. The complete subdivision is shown in Figure 9.

This structure provides the mesh with a kind of hierarchy. A vertex that has been created by cutting an edge in two is called "child vertex". The extremities of this edge are called "parent vertices".

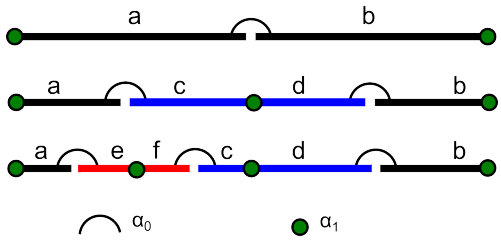

Figure 3: Edge refinement on a g-map (Level of black darts is 0. Blue stands for level 1 and red for level 2)

3.1 Definition

The definition of a dart must be extended. The notion of level is introduced, which corresponds to the number of refinements required to make the given dart appears. By default, the level of all initial darts is 0. The higher the level, the finer the resolution. Figure 3 shows the application of two successive refinement processes on an edge. Considering an initial level 0 for darts a and b, a first refinement process introduces new darts c and d with level 1. Another refinement gives new darts e and f with level 2. As shown, α_0 can link darts with different levels, but every α_1 links darts with the same level. The key point is that a new vertex is introduced only when an edge is cut. Thus, only α_0 supports resolution changes, the other involutions are kept unchanged or new ones are added between new darts only. Note that in the remainder of this

article, α_0 are no longer explicitly represented in figures in order to make these more clear.

Figure 4 shows dart levels while applying a Loop subdivision to a triangle. New inner darts also get level 1. On the right part, another subdivision is applied to one of the resulting triangles. Remark that α_1 and α_2 always link darts with the same level.

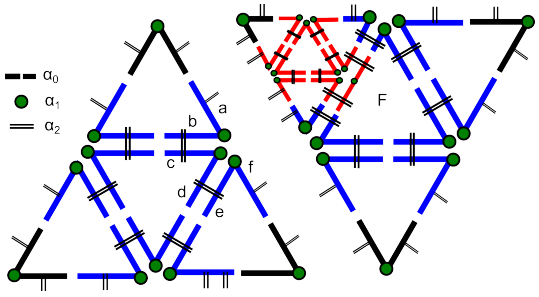

Figure 4: Face subdivision on a face of a 2-g-map (the level of black darts is 0, blue stands for level 1 and red for level 2)

Levels are also supplied to i-cells. This leads to the following theorem:

Theorem 1. *Any α_i, where $i > 0$, links two darts with the same level.*

Proof. : When a refinement is applied, only α_0 are modified for any already-existing darts and no changes are applied to other α_i. *Ad absurdio*, let's suppose that there exists an α_i linking two darts with different levels. That means that, at some coarse level, one of the darts would not exist yet and the other one would be i-free. This status cannot be changed in further operations. □

Since vertex orbit does not contain α_0, that means that all the darts in a vertex orbit have the same dart level. This is called *vertex level*. Note that this property appears as an invariant that topological operations should guarantee, which helps validating their implementation. Since a refinement cuts edges and provides the new ones with new darts, the definition of *edge level* is immediate:

Definition 2 (Edge level)**.** The level of an edge orbit is defined as the maximum of the levels of the darts composing this orbit.

Faces and volumes are also provided with a level. They also correspond to the number of subdivisions necessary to create them. However, these levels cannot be computed by just considering the level of the darts included in the orbit. In Figure 5, two triangles have been subdivided first, and one of the resulting 1-level face has been further refined. It can be seen, for example on the face pointed at by an arrow, that a face level is neither the maximum of the levels of its darts nor the minimum. As a consequence, face and volume levels are explicitly coded in the structure.

In the remainder of this paper, $level(d)$ denotes the level of a dart d. $level^i(d)$ is the level of any i-cell supported by d. For instance, $level^1(d)$ corresponds to the *edge level* of d. Note that at any time, all the darts of a face have the same $level^2$ and the darts of a volume have the same $level^3$ (the level is related to the embedding of the i-cell).

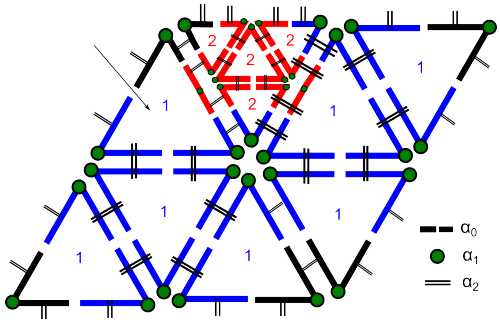

Figure 5: Several refinements applied to two initial 0-level faces (black stands for level 0, blue stands for level 1 and red for 2)

3.2 Queries

The subdivision of an i-cell implies that, on its boundary, the common $(i-1)$-cell with each of its adjacent i-cells is subdivided. For instance, when a tetrahedron is subdivided, its adjacent tetrahedra, if non-refined, see one of their faces divided by a Loop scheme. This issue in $2D$ can be seen in Figure 4. Since one of its adjacent triangles has been refined, face F is no longer a "topological" triangle, but a quad. In such cases, it is important to find a way to navigate in the structure. More precisely, the vertices of a non-refined original triangle should be found while ignoring the vertices introduced in the middle of its edges (the level of such vertices is higher than the level of the face). To allow a proper navigation, an α_0 which corresponds to the level of the triangle, and not the level of the edge, is needed. The same problem appears in $3D$ when edges and/or faces of a tetrahedron have been refined, and an α_0 which corresponds to its volume level is needed.

Instead of tracing the different α_0 for each level (after a cutting, such a structure does not remain consistent), an algorithm which finds the dart linked by α_0 for a given level is preferred. It is denoted as α_0^{lev}. This algorithm is recursive: To get α_0^i, it is necessary to get α_0^{i+1}. Naturally, this is not useful if the support edge has not been further refined, since level $i+1$ does not exist. In such a case, α_0 gives the answer and allows the end of recursion. The main idea is to find the adjacent dart at level $i+1$ in order to get the vertex in the middle of the edge at level i. Then, in this vertex orbit, the dart that "extends" this edge must be found. If the adjacent face is not refined, only applying α_1 is sufficient. On the contrary, due to the Loop scheme, the combination

$(\alpha_1 \circ \alpha_2)^{(2)} \circ \alpha_1$ is used (see how dart f can be found starting from dart a in Figure 4). When the dart that extends the support edge is found, another call to find its α_0^{i+1} allows the algorithm to find the searched dart. As an example, in Figure 6, let us assume that $\alpha_0^0(a)$, that is b, must be found. This requires to find $\alpha_0^1(a) = c$. Applying α_1 to c (or a combination of α_1 and α_2 if the adjacent face is refined), it is possible to find d. Applying another α_0^1 allows one to find b. During recursion, c is found by first searching for e, then computing f and then c. If the face supported by the request dart has not been refined, no recursion is needed and a simple walk alternating α_0 and α_1 is sufficient to find the desired dart (the level of this dart is at most i, so the algorithm goes on while such a level is not found). That means that in Figure 6, it is also possible to find b by just walking through e, f, c and d successively, b is the first met dart with a level 0 less than or equal to the request level 0. The overall algorithm can be found in appendix.

Note a strong hypothesis in $3D$: If α_0^{lev} is required, the level of the support volume is supposed to be less than or equal to lev. If its level is greater, finding α_0^{lev} would require to walk through some sub-volumes. However, the number of these sub-volumes cannot be known easily (4 or 5 sub-volumes), since it depends on how the refinement has been applied. Fortunately, in treatments described below, it is always possible to work with a non-refined volume.

Another important point is that a very similar algorithm provides a coverage of all the darts composing an edge at a given level.

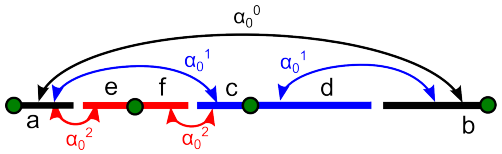

Figure 6: Different α_0 queries depending on a given level (black stands for level 0. Blue stands for level 1 and red for level 2)

Finally, a last query is mandatory: When an edge has been split, the newly-created vertex may need to find its parents. A wrong approach consists in memorizing the parent vertices once and for all when a child vertex is created. When topological changes occur, ambiguities might appear: Any parent vertex can split into two vertices and each of its children must update its parent accordingly to match the correct one. This problem is illustrated in Figure 7 where a 2-unsewing is applied. It is worth noting that multiple potential children can be concerned, if multiple refinements have been applied. Indeed, in $3D$, this parent link problem concerns any introduced vertex on the boundary of a face adjacent to a 3-unsewn face. Another approach consists in memorizing, for each new dart of a child vertex, the dart of the corresponding extremity of the cut edge (the dart that is 0-sewn at the level of the new dart). This approach is quite memory consuming: It requires to memorize a different dart for only some of new darts (those coming from an edge cut) and moreover does not take benefit

from the previously-defined α_0^{lev} query. Therefore, a way to compute parents in order to correct parent links after a topological operation is described below.

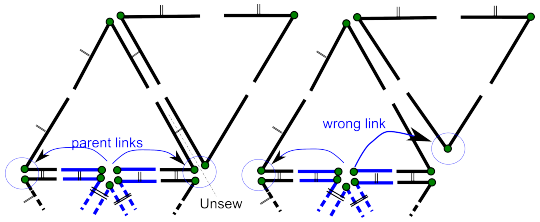

Figure 7: Example of parent inconsistencies appearing after a 2-unsewing

In subsection 4.5, the request for parents appears really useful whenever one of the surrounding volumes of a child vertex, in $3D$, has not been refined up to the level of this vertex. Under this assumption, the algorithm first selects a volume that has not been refined, that is, finds a dart dc in the child vertex orbit such that $level^3(dc) < level(dc)$. Let us call lev the level of dc, which by the way is also the level of the vertex child, considering theorem 1. In this volume (*i.e.* the use of α_3 is prohibited), the edge that has been cut to create the given child vertex must be found. Originally, this edge was composed of darts with level strictly less than the level where the child vertex was created. Other edges that have been connected to this child vertex have appeared at higher levels and include only darts with a level greater than or equal to lev. In Figure 4, it can be remarked that in the vertex orbit a, b, c, d, e, f, only a and f are 0-sewn (at level 1, the level of this vertex) with darts with lower levels. These darts give the parents of this vertex and they indeed verify $\alpha_0^0(a) = f$. These lower dart levels is the main property that allows the algorithm to find parent vertices.

More formally, each dart d in the orbit $< \alpha_1, \alpha_2 > (dc)$ is considered. $d1 = \alpha_0^{lev}(d)$ is computed and its level is compared to lev. If $level(d1) < level(dc)$, a parent dart $d1$ has been found. The other parent can either be found by carrying on the same algorithm and find another dart with the same property as $d1$ but not in the same vertex orbit, or, more directly, by computing $\alpha_0^{lev-1}(d1)$. Note that, in $2D$, the algorithm works the same way (but, contrary to the $3D$ case, adjacent faces can be refined).

3.3 Discussion

It is often argued that generalised maps are not optimal in terms of memory size. Indeed, more condensed models such as winged-edges or half-edges exist. Nevertheless they are restricted to $2D$. Dual combinatorial maps appear as an extension of half-edges to other dimensions and deserve to be examined carefully. Kraemer et al. [KCB09] and later, Untereiner et al. [UCB13] proposed a multiresolution topological model based on combinatorial maps and adapted the same Loop subdivision scheme, and others, to their framework. It can be pointed out that using their model, every permutation and involution is modified

after a subdivision is applied. These links must be memorised for each dart, as it is tricky to design queries similar to our α_0^{lev}-query. If combinatorial maps require half as many darts as generalised-maps, all the saved memory is lost to keep permutations and involutions at all levels. In our adaptive approach, generalised maps are truly a compromise, since the resolution change is locally defined only on one involution, α_0, that our approach avoids to memorise for each level.

4 Hierarchical Subdivision

This section describes the used refinement process. Only the $3D$ case is considered, for $2D$ is a sub-case. To subdivide a volume, it is necessary to subdivide its faces first, which implies the subdivision of their edges. When a face is refined, this operation concerns the one or two volumes it is connected with. Edge refinement has an impact on all the faces (and their adjacent tetrahedra) surrounding the edge. This implies that, for any edge or face subdivision request, it is first checked that such a subdivision has not already been applied, during a previous refinement of any adjacent cell. Before applying a subdivision, if the level of the cell to subdivide is greater than or equal to the requested level, nothing has to be done, except finding some darts representing the sub-elements resulting from the subdivision, when they are needed by the caller of the subdivision.

For the sake of clarity, the different refinement processes are presented in order of increasing dimension. The end of this section tackles the modification of embedding after the refinement process and deals with the interfaces between volumes (or faces) with different refinement levels (so-called "T-junctions").

4.1 Edge subdivision

Edge subdivision process is described in Figure 3. It basically relies on vertex insertion, a classical operation in topology [BDSM08]. In short, this consists in first creating "inner darts" linked by α_1 and connected to the original darts of a given edge by α_0. Note that if this edge belongs to multiple faces linked by α_2 and, possibly volumes linked by α_3, this operation is done for each edge common with adjacent faces and volumes. α_2 and α_3 links must be positioned in an isomorphic way as the original darts of the cut edge to ensure the quasi-manifold condition stated in definition 1. The level of any new edge is set to the subdivision level. The face and volume levels for new darts are easily obtained since they are the same as the one of their 0-sewn dart. Note that even if new darts belong to the same vertex orbit, they are likely to have different face and volume levels.

4.2 Face subdivision

Figure 8 shows the overall process of face refinement. After checking that the given face is not already subdivided at the requested level, its three edges must

be subdivided. Note that the α_0^{lev} query described in subsection 3.2 is of interest if the edges have already been subdivided, in order to find the three vertices of the triangle to subdivide. For each vertex, the algorithm gets the darts resulting from the edge subdivision and linked only by α_0 to the vertex orbit. By means of these two darts, a classical face cutting is applied [BDSM08] which consists in introducing a new edge that is 1-sewn to these darts. The two resulting faces are 2-sewn along the newly created edge. If the cut face is not 3-free, the face of the adjacent volume is transformed with the same process and the corresponding new darts of each face are 3-sewn.

The level of all new darts is set to the level of subdivision (see Figure 4). Their volume levels are duplicated from originally-existing darts of the subdivided face (note that the levels of the two volumes adjacent to this face can be different). The face level of new and existing darts for the four newly-created faces are set to the subdivision level, on both sides of the original face.

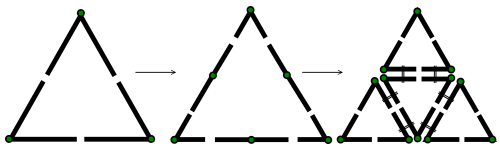

Figure 8: Steps of face refinement

4.3 Volume subdivision

Volume subdivision is presented in Figure 9. Volume subdivision does not require any volume level checks, it is applied whatever the volume level of any given tetrahedron. First, if needed, its four adjacent faces are subdivided at the needed level. Note that this requires no more than one subdivision for each face. The first step consists in cutting the volume around each vertex by inserting a face. An "edge path" must be defined which includes the edges resulting from the subdivision of the tetrahedron's faces around each vertex. These edges become the edges of the new inserted face inside the original tetrahedron. Thus, four tetrahedra, one for each vertex, are built. Since edges come from face subdivision, new darts of each inserted face are only duplicates of existing darts. Their level corresponds to the subdivision level, as well as their face level and volume level.

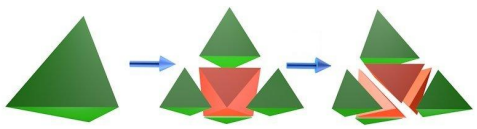

Figure 9: Volume subdivision

A central octahedron appears. The same subdivision scheme as [FDA02] is

used. It consists in (1) cutting the octahedron into two pyramids by following some of the edges resulting from the subdivision of its faces, (2) cutting the (unique) base of the pyramids in two along its diagonal and (3) cutting each pyramid into two tetrahedra using its apex and the above-mentioned diagonal. All new darts have a level, a face level and a volume level set to the subdivision level. All the volume levels of the existing darts of the original tetrahedron must also be updated.

4.4 Mechanical adaptation

When a refinement process is applied to a tetrahedron, new vertices, edges, faces and volumes are created. Since these cells embed mechanical properties, these must be added or adapted in a consistent way. New mechanical nodes are introduced in the middle of the edges of the initial tetrahedron. A new mass distribution is to be computed. After a subdivision, all the tetrahedra around any new nodes resulting from edge cuts must be covered to give their mass contribution. Note also that the mass contribution of the initial tetrahedron must be reduced by a factor 8 for its four vertices. Depending on the used deformation laws, new edges and/or new volumes must be mechanically embedded. During edge refinement, when a spring with stiffness k is cut, the stiffness of the two resulting edges is $2k$. This avoid instabilities since the new couple of springs is equivalent to the original one. More generally, for edges created from scratch, it is considered that $l_0 \times k$ is a constant of the body, where l_0 is the rest length of the spring and k its stiffness. The final stiffness is obtained by summing the stiffness brought by each volume surrounding the support edge. If FEM is used, the volume of each of the eight new tetrahedra must be computed, as well as their strain-displacement and rigidity matrices. Note that the cost of the overall process depends on the number of volumes around vertices (for mass computation) and edges (for stiffness computation if springs are used).

4.5 T-junctions

As explained previously, after a subdivision process, new vertices appear and change the topological type of surrounding faces and volumes. Triangles become quads, tetrahedra become heptahedra (when one face is transformed into four ones), and so on, as shown in Figure 10. When all adjacent cells are refined, a triangle can become an hexagon, and, in the same way, a tetrahedron becomes at most a polyhedron with 16 faces. More complex faces and volumes are obtained when higher level refinements are applied. However, in any case, such faces/volumes must remain triangles/tetrahedra.

To overcome this problem, a first approach consists in providing specific subdivisions in order to keep a conformal mesh. In [FDA02], different subdivision schemes can be found. We are interested in two of them:

- the one that cuts a tetrahedron so that its common face with another refined tetrahedron is subdivided using a Loop scheme,

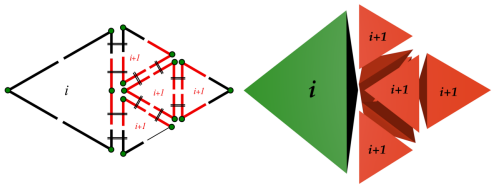

Figure 10: Examples of T-junctions in $2D$ (left) and in $3D$ (right)

- the one that cuts a tetrahedron in two because one of its edge has been subdivided.

These transformations are rather straightforward in our model. They rely on the same operations as previously, that is, cutting a face by inserting an edge and cutting a volume by inserting a face along an edge path. Note that such a subdivision implies to change the level of new volumes (see below the constrained vertex condition). However, this approach has a real drawback. It can be seen as a partial subdivision that prevents any further complete subdivision (as described in previous subsections). Instead, we rely on the approach in [HPH96, DMG05] where constraints are added in order to keep some vertices in the middle of their original edge. Such vertices are called "constrained vertices" in the remainder of the article (where Hutchinson *et al.* prefer "inactive").

First, the following definition is proposed: A vertex is constrained if at least one of its surrounding volumes (faces in $2D$) has a level less than its own level. More formally, considering a g-map in dimension n:

Definition 3 (constrained vertex)**.**
$constrained(dart) \equiv \exists d \in vertex\_orbit(dart)$ such that $level^n(d) < level(dart)$.

A vertex is free if it is not constrained. Figure 11 shows examples of constrained and free vertices in the $2D$ case. When a vertex is created during a subdivision, it gets mechanical properties, such as mass for instance. However, until it is free, it is not simulated. Instead, its position is computed as the middle of its parents. Note that one or both parents can themselves be constrained, so this process is recursive. Forces can be applied on constrained vertices (at least gravity, but also collision for instance). Half of any such forces is reported for each parent. This is also a recursive algorithm in case parents are constrained as well. After any topological change, the constrained condition must be checked for any implied constrained vertex. Some vertices can indeed be freed, as they are no longer adjacent to any lower-level volume (resp. face in $2D$). If a vertex is freed, it must be included in the simulation process by adding new degrees of freedom to the system.

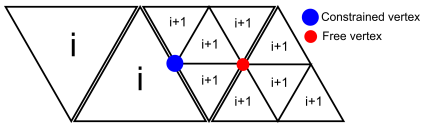

Figure 11: Example of a constrained vertex (in blue) and a free vertex (in red)

5 Topological modifications

It has been shown in [MDS10] that cutting and tear mainly rely on 3-unsewing (2-unsewing in $2D$), as well as matter destruction that requires first to unsew the faces of a volume in order to isolate and delete it. The main problem is that 3-unsewing or "face splitting" must be processed in a consistent way with the adaptive resolution. In $2D$, such modifications rely on 2-unsewing or "edge splitting". This section mainly focuses on the $3D$ case, since $2D$ appears more simple to handle. Nevertheless, the text is punctuated with elements related to $2D$ case to show the high correlation between both dimensions.

5.1 Topological face/edge splitting

Basically, face splitting consists in walking through the darts of the face ($< \alpha_0, \alpha_1 >$ orbit) and 3-unsewing them (resp., in $2D$, walking through the edge orbit and 2-unsewing the darts). A problem might appear if one of the vertices of the split face (resp. edge) is constrained. Figure 12 shows this problem in $2D$. In part (a), a constrained vertex has to remain in the middle of its support edge, since one of the adjacent faces is not enough refined. The obtained behaviour is quite weird, because one resulting edge can move while the other is constrained. To avoid such cases, face (resp. edge) splitting must be applied at the level of the least refined adjacent volume (resp. face), as shown in part (b). Thereafter, unsewing must be applied for all the sub-faces (resp. sub-edges). In $2D$, this is quite straightforward using different α_0^{lev} queries. In $3D$, a complete walk through all the darts of sub-faces is necessary. A recursive treatment of the Loop subdivision of the face can be used, but can be rather inefficient (darts may be unsewn several times, so they have to be marked to be treated only once, and unmarked at the end of the algorithm). Another approach consists in 2-unsewing the face to be split from the rest of the map (that is no longer a formal g-map, since a "cap" has been removed) and 3-unsewing all darts in the cap, by walking through $< \alpha_0, \alpha_1, \alpha_2 >$ orbit. Sewing the cap again by α_2 to the rest of the object is required to get a proper g-map again.

Face or edge splitting can imply vertex splitting(s) (see Figure 12). Note that a constrained vertex can be freed after such a splitting, so the constraint condition must be checked.

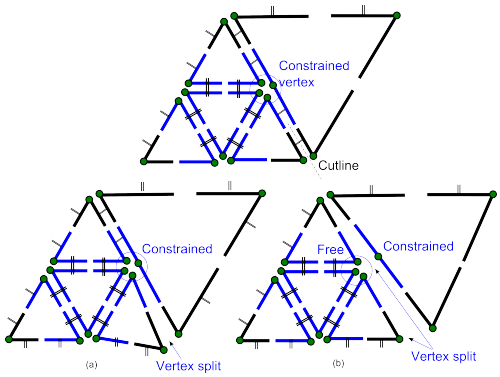

Figure 12: Example of vertex splitting in $2D$

5.2 Mechanical modifications

After a face splitting, edges and consequently vertices can be split. First, a splitting can be detected by tracing couples of darts before unsewing and checking if the darts of these couples belong to different orbits after modification. When an edge splits, and if springs are used, new stiffness must be computed by walking through all its surrounding volumes (both adjacent faces in $2D$). Basically, each edge of a volume (resp. face) is considered as a spring, so an edge is supposed to include several springs in parallel that are distributed in case of edge splitting. Note that edges of all sub-faces of a refined face are also subject to this process. If FEM is used, face splitting does not imply any change, even when adjacent cells are split.

The mass of any split vertex must be computed, again by walking through its surrounding volumes/faces and taking into account every mass contribution. The same remark as in section 4.4 applies: the cost of the overall process depends on the number of volumes surrounding vertices and edges.

6 Results

Subdivision and face splitting have been implemented to allow resolution refinement before applying cutting or matter destruction. We used the MOKA library (freely available at `http://moka-modeller.sourceforge.net`) as 3-g-map engine. Figure 13 shows the result of successive randomized refinements and the robustness of the subdivision process. Figure 14 shows a matter destruction after refinement of a liver model. Figure 15 shows that modifications can be combined (a Blinn-Phong shading is applied on external faces to improve rendering). A video can be found at `http://vimeo.com/92312840`.

Performances have been measured on an Intel Core i7 at 2.7GHz. Two liver models have been used. The first one comes from the SOFA framework (freely available at `http://www.sofa-framework.org`) and includes 181 nodes, 914

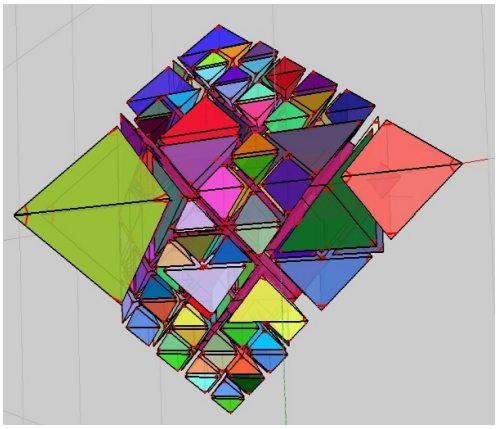

Figure 13: Example of several successive refinement operations

edges and 597 tetrahedra. The other one is more tessellated and includes 564 nodes, 2850 edges and 1856 tetrahedra. As explained in [MDS10], the use of a topological model does not significantly affect performances, provided that optimisation tables are computed to avoid long walks through the structure when no topological transformations occur.

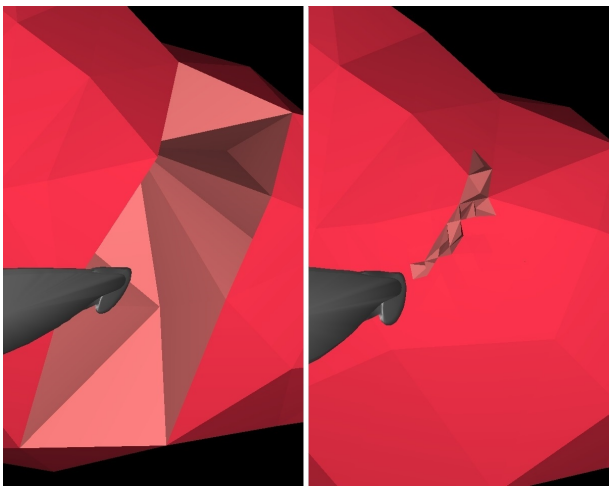

Figure 14: Examples of matter destruction on a coarse mesh (left) and a refined one (right)

Table 2 shows different average simulation time using both models. A simulation step includes force computation and Runge-Kutta 4 integration. Force computation depends on the chosen mechanical model: MSS, linear FEM or Nesme et al.'s QR-based co-rotationnal approach [NPF05]. Note that the tested

simulation did not take benefit from any parallelism on GPU.

Table 1: Computation times of a single simulation step for two models of liver (in ms)

Liver model	MSS	LINEAR FEM	QR FEM
SOFA Model	0.8	1.0	1.37
Complex model	2.2	3.7	5.0

Table 2 shows computation times of different topological modifications. These computation times include not only modifications of the topological model and its adaptive structure but also every mechanical properties update and detection of freed vertices. As shown in previous sections, all these updates necessitate various walks through the structure that require computation time.

Table 2: Computation times for several topological modifications (in ms)

Liver model	Face splitting	Tetrahedron removal	Refinement
SOFA Model	0.2	0.7	1.3
Complex model	0.4	1.1	1.3

Note that face splitting must be applied to several faces to be visible, so splitting enough faces to split a vertex can require a bit more than 1ms. This also explains why face splitting and consequently tetrahedron removal (that includes one, two or, more often, three face splittings) depends on the tessellation of the modified mesh (in particular the number of volumes surrounding vertices and edges). If adjacent cells (edges and more profitably faces) are already subdivided, a refinement can be less costly than shown in table 2. This is desirable since, most of the time, a subdivision process is applied to a tetrahedron and its surrounding tetrahedra to allow topological modifications as stated in section 5. This may concern more than a tenth of tetrahedra, so the overall refinement process may require between 10 and 20ms. This may appear costly compared to a simulation step. Nevertheless, these changes occur very rarely (several seconds can separate two successive changes) and, in practice, the missed simulation steps are acceptable.

Conclusion and prospects

In this paper, we have shown a topologically-based mechanical model, that allows a robust handling of modifications such as refinement, cutting, volume destruction, *etc.* We proposed a robust, mathematically specified, approach to deal with topological changes. This model behaves well and offers a real flexibility. Is is currently intended to implement this model into the SOFA framework.

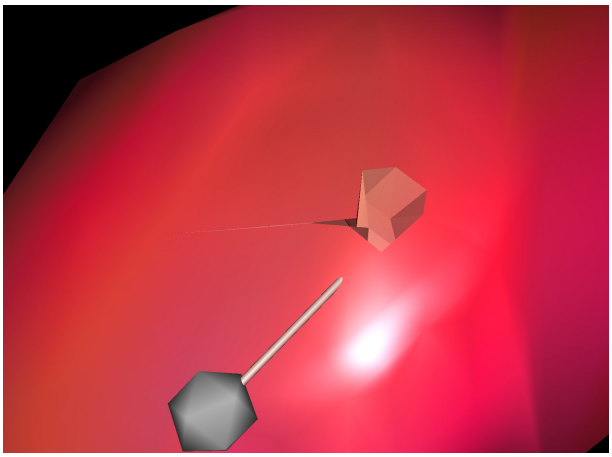

Figure 15: Combination of refinement, deletion and cutting

A GPU version could provide a faster simulation but only between topological changes that are computed on the CPU. In other words, that means that we extend our optimisation structure in order to export it to the graphics card memory.

Since the used topological model allows the representation of any kind of faces or volumes, an extension to hexahedra is absolutely possible. For this purpose, a dedicated refinement scheme must be chosen and new navigation processes must be designed, but no huge adaptation should be required. Another interesting topological modification deals with coarsening. It is not sure that the proposed structure allows the location of zones that can be coarsened. Tracing all the resolutions of the body through a hierarchical model seems mandatory in this case. However, keeping a consistent hierarchical structure that allows topology modifications is a real issue, therefore more studies are needed.

Addendum

As stated at the beginning of this document, our work is very similar to [FZDJ14]. In this section, we try to highlight some fundamental differences. First, Fléchon et al. use combinatorial maps, but in practice, combinatorial maps appear as optimizations of g-maps, so any approach using combinatorial can be potentially applied to g-maps and vice versa. In their work, they define hierarchical darts that is, darts that exist before a subdivision is applied and allow to identify a cell at a coarse level after several subdivisions. In our model, these darts correspond to darts of the considered cell with a level less than or equal to the level of the cell. Besides, some of their algorithms that need to walk through a face of a volume may require to walk through darts on the adjacent volume [Flé14], since no α_0^i-equivalent request is proposed. This suppose that in their model,

any subdivided face must be sewn to another volume by this face. If a cutting is applied on such a face, any unnecessary subdivision (that is, where the adjacent volume is not subdivided) should be suppressed. In our approach, this simplification step is not mandatory.

We can also mention the work of Untereiner et al. [UKCB15] that provides a convenient way to implement and walk through multiresolution models. They use combinatorial maps and, in the same philosophy as us, provide an algorithmic way to find permutation and involutions at any given resolution. Note that, although only primal subdivisions can be applied (that is, subdivisions that rely on edge cuts), combinatorial maps imply to implement a level-based permutation, and define any involution, for a given level, by relying on the level-based permutation (in other word, they cannot only focus on level-based α_0). Their model has nevertheless the great advantage to be completely general and extensible, not limited to only tetrahedra (they can handle hexahedra as well). This model could surely be used to get similar results as ours.

Finally, let us mention two more recent papers. Koschier et al. [KLB14] provide an adaptive refinement in 5 steps for tetrahedral models dedicated to fracture and Paulus et al. [PUC+15] rely on a mesh modification that keep the number of nodes constant while providing a better fitting of a cutting path.

Acknowledgment

We would like to thank Guillaume Damiand and Sébastien Horna for their help in using MOKA. We also thank Christian Duriez and Jérémie Dequidt for helping us implement FEM.

Appendix

This section shows the algorithm that computes the α_0 link at a given level. The underlying volume is supposed not to be refined more than this level.

```
1: function α_0^lev(dart:Dart) : Dart
2:     // Dart must exist at this level, so has a lower level
3:     assert(lev≥ level(dart))
4:     assert(lev≥ level^3(dart)) // Volume must not have been refined
5:     assert(lev≤ level^1(dart)) // Request level exists ?
6:     Dart d
7:     if lev= level^1(dart) then
8:         return α_0(dart)
9:     end if
10:    if lev< level^2(dart) then
11:        // the face is cut, we have to walk through the
12:        // 3 sub-faces to find the extension of the edge
13:        d ← α_0^(lev+1)(dart)
14:        d ← α_1 ∘ α_2 ∘ α_1 ∘ α_2 ∘ α_1(d)
15:        return α_0^(lev+1)(d)
16:    else
17:        // face not subdivided, we just walk along the edge
18:        d ← α_0(dart)
19:        while level(d)>lev do
20:            d ← α_0 ∘ α_1(d)
21:        end while
22:        return d
23:    end if
24: end function
```

References

[Bat82] K. Bathe. *Finite Element Procedures in Engineering Analysis*. Prentice-Hall, 1982.

[BDSM08] Mehdi Baba-Ali, Guillaume Damiand, Xavier Skapin, and David Marcheix. Insertion and expansion operations for n-dimensional generalized maps. In *Discrete Geometry and Computer Imagery*, volume 4992 of *Lecture Notes in Computer Science*, pages 141–152, Lyon, France, 2008. Springer Verlag.

[BG00] Daniel Bielser and Markus H. Gross. Interactive simulation of surgical cuts. In *Pacific Graphics*, pages 116–125, 2000.

[BGTG03] Daniel Bielser, Pascal Glardon, Matthias Teschner, and Marlus Gross. A state machine for real-time cutting of tetrahedral meshes. In *Pacific Graphics*, pages 277–386, Canemore, Canada, 2003.

[Bri89] E. Brisson. Representing geometric structures in d dimensions: topology and order. In *Annual symposium on Computational geometry*, pages 218–227, 1989.

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

[CMQ02] Y.-S. Chang, K. T. McDonnell, and H. QIN. A new solid subdivision scheme based on box splines. In *ACM symposium on Solid modeling and applications*, pages 226–233, 2002.

[DDCB01] G. Debunne, M. Desbrun, M.P. Cani, and A.H. Barr. Dynamic real-time deformations using space and time adaptive sampling. In *SIGGRAPH 2001 Conference Proceedings*, Computer Graphics annual conference series, Los Angeles, August 2001.

[DGW11] C. Dick, J. Georgii, and R. Westermann. A hexahedral multigrid approach for simulating cuts in deformable objects. *IEEE Trans. on Visualization and Computer Graphics*, 17(11):1663–1675, 2011.

[DMG05] J. Dequidt, D. Marchal, and L. Grisoni. Time critical animation of deformable solids. *Computer Animation and Virtual Worlds*, 16(3–4):177–187, July 2005.

[FDA02] C. Forest, H. Delingette, and N. Ayache. Cutting simulation in manifold volumetric meshes. In *Medical Image Computing and Computer Assisted Intervention*, pages 235–244, 2002.

[FDA05] 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, April 2005.

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

[Flé14] E. Fléchon. *Définition d'un modèle unifié pour la simulation physique adaptative avec changements topologiques.* PhD thesis, University of Lyon 1, 2014.

[FZDJ13] E. Fléchon, F. Zara, G. Damiand, and F. Jaillet. A generic topological framework for physical simulation. In *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 104–113, Plzen, Czech Republic, 2013.

[FZDJ14] E. Fléchon, F. Zara, G. Damiand, and F. Jaillet. A unified topological-physical model for adaptive refinement. In *Eurographics workshop on Virtual Reality Interactions and Physical Simulation*, pages 39–48, Bremen, Germany, September 2014.

[GCMS00] Fabio Ganovelli, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A multiresolution model for soft objets supporting interactive cuts and lacerations. *Computer Graphics Forum*, 19(3):271–281, 2000.

[HPH96] Dave Hutchinson, Martin Preston, and Terry Hewitt. Adaptive refinement for mass/spring simulations. In *Eurographics workshop on Computer animation and simulation*, pages 31–45, Poitiers, France), 1996.

[KCB09] Pierre Kraemer, David Cazier, and Dominique Bechmann. Extension of half-edges for the representation of multiresolution subdivision surfaces. *The Visual Computer*, 25:149–163, January 2009.

[KLB14] Dan Koschier, Sebastian Lipponer, and Jan Bender. Adaptive tetrahedral meshes for brittle fracture simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2014.

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

[LJF+91] A. Luciani, S. Jimenez, J.L. Florens, C. Cadoz, and O. Raoult. Computational physics : a modeler simulator for animated physical objects. In *EUROGRAPHICS Workshop on Computer Animation and Simulation*, pages 425–437, Vienne, 1991.

[Loo87] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

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

[MDS10] P. Meseure, E. Darles, and X. Skapin. Topology-based physical simulation. In *Eurographics workshop on Virtual Reality Interactions and Physical Simulation*, pages 1–10, Copenhaguen, Denmark, November 2010.

[MG04] Matthias Müller and Markus Gross. Interactive virtual materials. In *Graphics Interface*, pages 239–246, London, Canada, May 2004.

[MK00] A. Mor and T. Kanade. Modifying soft tissue models: progressive cutting with minimal new element creation. In *Medical Image Computing and Computer Assisted Intervention*, pages 598–607, 2000.

[NPF05] Matthieu Nesme, Yohan Payan, and François Faure. Efficient, physically plausible finite elements. In *Eurographics*, Dublin, Ireland, 2005.

[NTB+91] A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney. Animation of fracture by physical modeling. *The Visual Computer*, 7:210–219, 1991.

[NvdS00] H.W. Nienhuys and A.F. van der Stappen. Combining finite element deformation with cutting for surgery simulations. In *EUROGRAPHICS (short presentations)*, pages 43–51, August 2000.

[NvdS03] H.W. Nienhuys and A.F. van der Stappen. A delaunay approach to interactive cutting in triangulated surfaces. In *Int. Workshop on Algorithmic Foundations of Robotics*, volume 7, pages 113–130. Springer Verlag, 2003.

[OBH02] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. *Transaction on Graphics (SIGGRAPH)*, pages 291–294, August 2002.

[Pro95] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics Interface*, pages 147–154, May 1995.

[PUC+15] Christoph Paulus, Lionel Untereiner, Hadrien Courtecuisse, Stéphane Cotin, and David Cazier. Virtual cutting of deformable objects based on efficient topological operations. *Visual Computer*, pages 1–11, May 2015.

[SDF07] E. Sifakis, K. Der, and R. Fedkiw. Arbitrary cutting of deformable tetrahedralized objects. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 73–80, San Diego, 2007.

[SHGS06] Denis Steinemann, Matthias Harders, Marlus Gross, and Gabor Szekely. Hybrid cutting of deformable solids. In *IEEE Conf. on Virtual Reality*, 2006.

[SHS01] D. Serby, M. Harders, and C. Székely. A new approach to cutting into finite element models. In *Medical Image Computing and Computer Assisted Intervention*, pages 425–433, 2001.

[SOG06] D. Steinemann, M. A. Otaduy, and M. Gross. Fast arbitrary splitting of deforming objects. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 63–72, 2006.

[TF88] D. Terzopoulos and K. Fleisher. Modeling inelastic deformation : Viscoelasticity, plasticity, fracture. *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, 22(4):269–278, August 1988.

[TPBF87] D. Terzopoulos, J. Platt, A. Barr, and K. Fleisher. Elastically deformable models. *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21(4):205–214, July 1987.

[TSSB+05] J. Teran, E. Sifakis, S. Salinas-Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics*, 11:317–328, May 2005.

[UCB13] Lionel Untereiner, David Cazier, and Dominique Bechmann. n-dimensional multiresolution representation of subdivision meshes with arbitrary topology. *Graphical Models*, 75(5):231–246, 2013.

[UKCB15] Lionel Untereiner, Pierre Kraemer, David Cazier, and Dominique Bechmann. Cph: a compact representation for hierarchical meshes generated by primal refinement. *to appear in Computer Graphics Forum*, 2015.

[WWD14] Jun Wu, Rüdiger Westermann, and Christian Dick. Physically-based simulation of cuts in deformable bodies: A survey. In *Eurographics STARs*, pages 1–19, Strasbourg, France, 2014.