A General Physical-topological Framework using Rule-based Language for Physical Simulation

Fatma Ben Salah, Hakim Belhaouari, Agnès Arnould and Philippe Meseure
University of Poitiers, Xlim Lab, UMR CNRS 7252, Futuroscope, France

Keywords: Physical Animation, Topologically-based Simulation, Rule-based Language, Prototyping.

Abstract: This paper presents a robust framework that combines a topological model (a generalized map or G-map) and a physical one to simulate deformable objects. The framework is general since it allows a general simulation of deformations (1) in different dimensions (2D or 3D), (2) with different types of meshes (triangular, rectangular, tetrahedral, hexahedral, and combinations of them...) and (3) physical models (mass/spring, linear FEM, co-rotational, mass/tensor). Any mechanical information is stored in the topological model and is used in combination with the neighboring relations to compute the equation of motions. To design this model, we have used JERBOA, a rule-based language relying on graph transformations to handle G-maps. This tool has been helpful to build and test different physical models in a little time.

1 INTRODUCTION

Physically-based simulation have a great importance for various computer graphics applications such as virtual reality, simulation of natural phenomena and video games. Since the early approaches of Terzopoulos et al. on simulating deformable objects (Terzopoulos et al., 1987), several models have focused on simulating realistic behavior of deformable objects. These models rely on discrete mechanics (mass/spring systems (Provot, 1995)) or continuous mechanics (Finite Element Method (O'Brien and Hodgins, 1999)), mass/tensor systems (Cotin et al., 2000), Finite Differences (Terzopoulos et al., 1987)).

Physical models can undergo topological modifications whereas they are based on a rudimentary topological structure. This usually implies heavy processes to ensure that the mesh remains manifold (Forest et al., 2005). Yet, it is possible to preserve the manifold property of a mesh by using more robust and appropriate topological models (Damiand and Lienhardt, 2014). The consistency is therefore guaranteed after topological modifications.

Meseure et al. (Meseure et al., 2010) proposed to combine a physical model with a topological one. Their approach is specific to tetrahedral meshes and mass/spring systems. Thereafter, Fléchon et al. (Fléchon et al., 2013) extend the use of topological models to simulate hexahedral (3D) or rectangular (2D) meshes with mass/spring systems. These approaches show the effectiveness of using a topological model in physically-based animation, particularly for simulating transformations such as tearing, cuttings, etc. However, their solution under-exploit the capabilities of the topological model to embed any kind of information. For instance, shear springs are stored in an additional structure which appears redundant with the topological model (and can potentially result in an inconsistent model association). Recently, Golec et al. (Golec et al., 2015) have proposed a generic approach to simulate both tetrahedral and hexahedral meshes using Mass/spring or Mass/tensor physical models. Their method unfortunately still requires an additional structure to store shear springs and tensors. This structure includes topological information that must be kept consistent with the topological model. This is a real issue in case of complex topological transformations.

This paper proposes several contributions:

- formalization of the physical modeling and animation of deformable meshed objects using an appropriate topological model (namely generalized maps or G-maps),
- generalization of the modeling and the behavior computation with respect to (1) the dimension, (2) the type of mesh elements and (3) the constitutive laws,
- fast prototyping using a formal approach based on rules of graph transformations.

Ben Salah F., Belhaouari H., Arnould A. and Meseure P.
A General Physical-topological Framework using Rule-based Language for Physical Simulation.
DOI: 10.5220/0006119802200227
In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017)*, pages 220-227
ISBN: 978-989-758-224-0

It is organized as follows. Section 2 presents the related work. In Section 3, the main basis of prototyping with a rule-based language using the JERBOA software (Belhaouari et al., 2014) is detailed. In Section 4, the physical simulation of 2D and 3D objects with different types of meshes and physical models using rule-based language are described. Finally, implementation details are given in Section 5.

2 PREVIOUS WORK

Several previous studies have shown the advantages of using a topological model for simulation. Meseure et al. (Meseure et al., 2010) use G-maps to simulate the deformation and the topological changes of soft bodies. Their approach is restricted to tetrahedral meshes and mass/spring systems. Moreover, they have used only stretching springs, associated to edges. Nevertheless, due to the G-maps engine used, they required external arrays to allow a fast access to cells (vertices, edges, volumes...). These arrays must be updated after any topological change.

Fléchon et al. (Fléchon et al., 2013) have used a derivation of combinatorial maps, namely 3D Linear Cell Complex (LCC) as a topological model to simulate mass/spring systems based on 2D rectangular and 3D hexahedral meshes. In this model, stretching springs are also associated to edges. Unfortunately, shear springs (placed along diagonals of each element) require an external structure. Golec et al. (Golec et al., 2015) extend this model to allow the simulation of 3D tetrahedral and hexahedral meshes using not only mass/spring but also mass/tensor models. They claim not to rely on any additional structures but each face/volume is associated with an array of darts that allow the numbering of each vertex. Actually, this array implicitly includes neighboring relationships so appears as redundant with the LCC.

The real weakness of all these approaches is the use of external structures to store information required by the simulation. After a topological modification, the external structure can be rebuilt from scratch or be modified incrementally. The first solution guarantees correctness but is costly, while the second is a tedious process that can lead to inconsistencies. On the contrary, maps ensure their own consistency using constraints that can easily be verified.

It seems that the limitation of element types in (Meseure et al., 2010) and the use of external structures in (Fléchon et al., 2013) and (Golec et al., 2015) is due to an unsuitable modeling of interactions. Indeed, these approaches search for topological elements that could support both the origin of the forces (e.g. a link between two particles) and the vertices associated with the concerned particles. Concerning stretching springs, the solution is simple and immediate, since these topological elements are edges of the mesh. But the other kinds of springs do not directly correspond to topological elements (for instance, diagonals of faces). We therefore propose a new paradigm to model springs and other mechanical components in general, that could be called "topology-based force field". Force modeling process should focus only on the origin of each force and store the mechanical parameters on related topological elements. Thus, if these elements are modified by topological modifications, the resulting forces should be automatically altered. The (two or more) involved particles should therefore be identified from the storage place, while computing interaction forces.

To sum up, our method overcomes the limitation of the approaches listed above as it proposes:

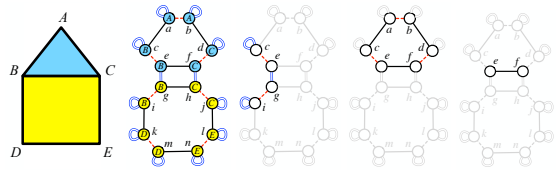
- a framework with multi-physical, multidimensional and multi-element simulation;
- a new paradigm to model interactions, by both considering their physical origin and identifying the (two or more) involved particles;
- a storage for **all** the mechanical information in the topological model (no other additional structure is necessary).

To simulate deformable bodies using a physical model and a topological one, a *rule-based language* is used which exploits graph transformations for prototyping. This language offers the opportunity to test the capability of our model to simulate different types of meshes in different dimensions with different physical models and different properties (homogeneity, isotropy) in a little time. Note that a lot of languages are dedicated to physical modeling, such as (Kjolstad et al., 2016) among others. These languages often propose to describe the simulated objects, including their topology, but contrary to our approach, they do not aim at programming the simulation itself nor do they allow the design of new interactions.

3 PROTOTYPING USING RULE BASED-LANGUAGE

The JERBOA rule-based language (Belhaouari et al., 2014) is dedicated to geometric modeling. It is based on the topological model of G-maps (Damiand and Lienhardt, 2014). The representation of an object using a G-map is defined from its successive subdivision into topological cells (vertices, edges, faces, volumes, etc.) and appears as a graph (Figure 1b). The

nodes of this graph are called *darts*. Arcs represent the adjacency relationships between cells and are labeled by their dimension. In the following figures, black arcs stand for 0-dimensional, red dotted arcs for 1-dimensional and blue double arcs for 2-dimensional links.



(a) (b) (c) (d) (e)

Figure 1: Decomposition of a 2D object and examples of orbits: (a) Object 2D, (b) G-map(connected component), (c) Vertex, (d) face and (e) half edge.

Topological cells (i.e. vertices, edges, faces, volumes) of an object are defined by sub-graphs called *orbits*. Figure 1 represents some orbits which correspond to cells of the object presented in Figure 1a. For example, vertex B of Figure 1a is defined by the sub-graph reachable from dart e using 1- and 2-links (that is, darts c, e, g, i and the links that connect them), noted orbit $\langle 1,2\rangle$(e) (Figure 1c). Similarly, edge BC is the orbit $\langle 0,2\rangle$(e) and face ABC is the orbit $\langle 0,1\rangle$(e) (see Figure 1d). We should note that the concept of *orbit* is not limited to cells and may identify less common entities, such as half-edges (Figure 1e), or connected components (Figure 1b).

To model an object, the topological structure of G-maps must be complemented by additional (geometric, colorimetric, mechanical...) information called *embeddings*. These data are attached to darts. For example, in Figure 1b, darts carry geometrical points (A, B, C, D and E) and colors (blue and yellow). Each embedding is characterized by the type of information (here a 2D point and a color) and the support orbit (here orbit $\langle 1,2\rangle$ for vertex information and $\langle 0,1\rangle$ for face information), but we can also associate information to any type of orbits like corner of face $\langle 1\rangle$. Besides, as shown in Figure 1b, all darts of a support orbit should share the same embedding.

A rule created with JERBOA has the form $L \rightarrow R$, where L (left part) is a sub-graph to be matched for the rule to apply and R (right part) a modification of this sub-graph, that includes expressions of embedding changes. JERBOA's rule editor automatically verifies and ensures the object consistency preservation. Besides, it allows a fast prototyping of geometric operations (some examples are given in Section 5).

4 RULE-BASED PHYSICAL SIMULATION

The modeling process is based on a 2D or 3D mesh which represents the structure of the object. It is modeled by a G-map, in which mechanical information is embedded. As stated in previous work, a correct embedding must be selected, that is, the most adapted topological place with respect to the physical origin of each property must be found. These properties include mass, ambient viscosity, etc. that relate to the physical model (studied in the next subsection) as well as any interaction model, whether it is discrete or continuous (studied in the two following subsections). The same process is always followed: (1) find the most adapted topological place to embed mechanical information, (2) from this place, define the adjacency relations to follow to find on which vertices/particles the force applies and (3) store the computed force in adapted sub-orbits of these vertices (for instance, corner orbit $\langle 1\rangle$ or extremity of edge $\langle 2,3\rangle$).

Based on this modeling, the simulation loop proceeds this way: (1) walk through all the orbits that embed interaction data and compute the applied forces, (2) walk through all the vertices and for each one, collect forces stored in its sub-orbits and compute the sum of forces applied to its particle and (3) integrate the equation of motion to find new velocity and positions of each particle.

4.1 Physical Model

The used simulation loop is the same for all physical models. This implies that every particle has a mass, as well as an acceleration, a velocity and a position that must be computed at each time step. These mechanical properties have to be stored in the most appropriate orbit. When the properties are related to a given cell, the orbit is found immediately. For instance, acceleration, position and velocity of a particle which are stored on its corresponding vertex (orbit $\langle 1,2,3\rangle$).

Considering mass, several solutions exist and ought to be discussed. Considering its physical origin, mass is supposed uniformly distributed over the surface or the volume of an homogeneous object. On the one hand, it seems natural to associate mass to the faces or volumes of the mesh. On the other hand, only models where mass is concentrated on particles each surrounding face/volume contributes provide a vertex with a mass) are considered so mass is stored in vertices in previous models (the mass of a particle is the sum of the contributions of each surrounding face/volume). As a compromise, the contribution of a face or a volume to each of its vertex is explic-

itly stored in our model. A face distributes its mass (evenly of not) to all its vertices, that is its sub-orbits $\langle 1 \rangle$ (known as "face corner"). The same way, a volume distributes mass to its sub-orbits $\langle 1,2 \rangle$ (known as "volume corner"). In both cases. Now the physical properties are defined, the interaction model that is applied to animate is presented in the following section.

4.2 Springs Model

Provot proposed three different types of springs, to control respectively stretching, shear and bending (Provot, 1995). This approach can be extended for 3D, where stretch springs correspond to edges of the mesh, and shear springs appear on diagonal of non triangular faces and/or non tetrahedral volumes.

As explained earlier, the process consists in finding the right orbit to embed spring information (stiffness, rest-length and possibly damping), defining a sub-graph to find the vertices implied in the interaction, computing it, and finally storing the computed forces in appropriate sub-orbits of these vertices. The sub-orbits where forces are stored can be found easily: The method always chooses the largest orbit included in the intersection of the sub-graph used to find concerned vertices (blue graph in following figures of this section) and the vertex orbit itself (orbit $\langle 1,2 \rangle$ in 2D and $\langle 1,2,3 \rangle$ in 3D, shown in red). It must be noted that it is delicate to find the right sub-graph for each type of springs since it must be chosen as generic as possible, so that it can be applied, if desired, on all types of faces (triangles, quads or others). Below, each type of spring is detailed.

4.2.1 Stretch Springs

Stretch springs connect two particles placed on vertices that are linked by an edge. In 2D, it seems convenient to associate spring information with edge orbits ($\langle 0,2 \rangle$) (as proposed in previous approaches). However, when an edge is adjacent to two faces, each face provides the edge with strength against stretching that can be represented as a spring. So, an embedding is created for any elementary spring (green sub-graph) associated to orbits $\langle 0 \rangle$ (face edges) as shown in Figure 2 and all these springs are summed for force computation. This gives the sub-graph shown in blue.

In Figure 2, the particles influenced by a stretch spring are the extremities of the support edge (shown in red). Forces are computed and stored in a sub-orbit of each vertex corresponding to the edge, that is $\langle 2 \rangle$. Here, faces are represented as a quad, but could include any number of vertices. This approach can be directly applied for 3D.

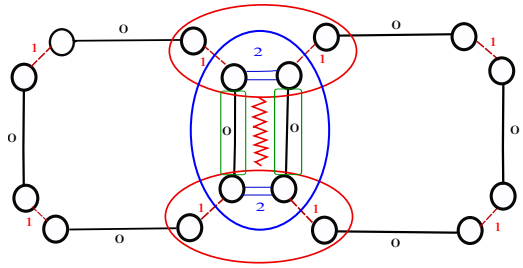
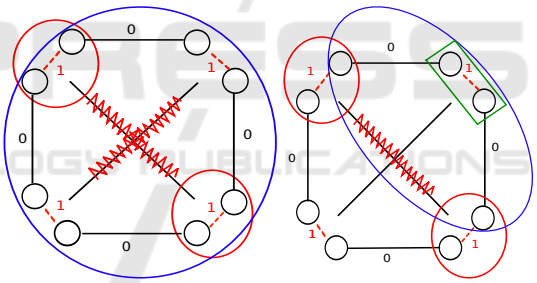


Figure 2: Force applied by stretch springs.

4.2.2 Shear Springs

Shear springs are supported by diagonals of a face, that is, segments that do not correspond to an orbit. Two main motivations can be considered to include shear springs: (1) because the face has to maintain its shape, so produces forces between each couple of opposite vertices or (2) because the angles of each corner of the face must be preserved, so a force must be applied to the non-connected extremities of any two adjacent edges. Since mechanical properties can be chosen freely for each couple of edges, this last approach is qualified as "anisotropic".



(a) Isotropic case (b) Anisotropic case

Figure 3: Force applied by shear springs.

In the isotropic model, a rectangular mesh must be used, since the same stiffness and rest-length is embedded in the face ($\langle 0,1 \rangle$) for both diagonals. The opposite vertices can be found by just walking through the face, so the face orbit $\langle 0,1 \rangle$ is the sub-graph needed to find vertices. Computed forces are stored in the intersection between the sub-graph and the vertex orbits, that is corners of the face ($\langle 1 \rangle$).

In the anisotropic model, elements can be of any shape or even any topology. The mechanical properties of a shear spring is stored in orbits $\langle 1 \rangle$ that corresponds to the articulation (the angle) between the two connected edges (shown in green). Since the intersection between the two articulated edges (blue sub-graph) and their non-connected extremities (shown in red) reduces to single darts, forces are stored in dart orbits $\langle \rangle$, as shown in Figure 3b. Note that this solu-

tion is compatible with triangles (although not useful, as a stretching spring already binds the same vertices).

In 3D, if shear springs are placed on the diagonals of hexahedra' faces, the same embeddings as in 2D still apply. If shear springs are put on the diagonals of a hexahedron, a new embedding is associated with its volume orbit ($\langle 0,1,2 \rangle$) and controls four similar springs. Note that, in this case, volumes should be rectangular cuboids. Here, computed forces are put in "corners" of volumes, that is orbits $\langle 1,2 \rangle$.

4.2.3 Bending Springs

Considering that bending must be handled by controlling the angle between any two adjacent faces, **angular** springs have been proposed to control bending ((Grinspun et al., 2003), among others), by constraining the angle between faces supported by the common edge of two adjacent faces. This angular spring can be stored in edge orbits $\langle 0,2 \rangle$ (green sub-graph in Figure 4).

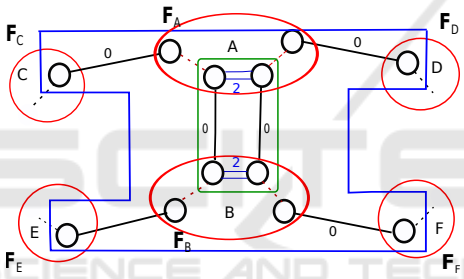


Figure 4: Force applied by an angular spring.

As shown in Figure 4, vertices of faces that are extremities of edges 1-linked to the articulation edge must be found (A, B, C, D, E and F) via an appropriate sub-graph (in blue), since they must undergo bending forces. Once calculated, these forces are stored in orbits dart $\langle\rangle$ (directly in the dart). Note that the chosen sub-graph is compatible with any type of faces, since the links between vertices C and E on the one hand, and D and F on the other hand, are not specified in the sub-graph. These vertices can be the merged (triangles), linked by an edge (quad), or linked by several edges (other types). Mixed meshes with different types of faces can be used as well.

To control bending, Provot introduced springs that connect second-neighbor vertices (Provot, 1995). These links control the angle between two edges both connected, via a vertex, to an edge shared by two faces. This remark gives the sub-graph, represented in blue in Figure 5. It seems convenient to embed such spring into the orbit $\langle 2 \rangle$, corresponding to the common edge and the articulation vertex at once (in green).

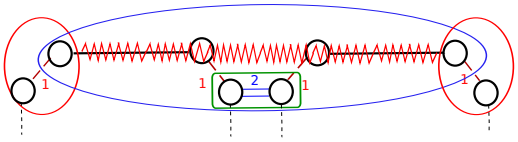


Figure 5: Force applied by a Provot's spring.

Computed forces are stored in dart orbit $\langle\rangle$. For the same reason as above, this modeling is compatible with triangles, even if two orbits $\langle 2 \rangle$ link the same couple of vertices. Mixed meshes can benefit from these springs as well.

Angular or Provot's springs aim at simulating the same effect but their implementations are different. Thus, both of them are proposed in our model, because provot's approach is more efficient in the case of a rectangular and triangular mesh, while the angular approach can potentially be generalized to other types of faces.

4.2.4 Force Computation

Once the vertices linked by a spring are identified, the force exerted by particle i on particle j is computed according to:

$$\mathbf{F_{ij}} = \left(-k_{ij}\left(\|\mathbf{x_j}-\mathbf{x_i}\|-l_{ij}\right)-\lambda_{ij}\left(\mathbf{v_j}-\mathbf{v_i}\right)\cdot\mathbf{u_{ij}}\right)\mathbf{u_{ij}} \quad (1)$$

where k_{ij} is the stiffness, l_{ij} the rest length, λ_{ij} the damping coefficient, $\mathbf{u_{ij}}$ the unitary vector from i to j, $\mathbf{x}$ the position and $\mathbf{v}$ the velocity of each particle.

To compute forces applied by angular springs on edge i, a formula quite similar to Grinspun et al.'s one (Grinspun et al., 2003) but simplified and extended to quads, is used for each involved vertex:

$$\mathbf{F_M} = \pm\frac{k_i \times \theta}{2 \times \|\mathbf{u}\times\mathbf{AM}\|^2} \times (\mathbf{u}\times\mathbf{AM}) \quad (2)$$

with θ is the angle between normals of the two faces (defined by the edge and the center of the quad), k the angular stiffness, $\mathbf{u}$ the unitary vector of the edge, $\mathbf{A}$ any vertex on the edge and $\mathbf{M}$ the vertex where the force is applied. When a vertex undergoes a force, the opposite of this force must distributed to vertices of the edge. Here, we adapt the method to quads by considering that $\mathbf{F_A} = -\mathbf{F_C} - \mathbf{F_D}$, (and similarly for $\mathbf{F_B}$) in figure 4.

The chosen embeddings in G-maps allow the modeling of different types of meshes including mixed ones, in 2D and 3D using the same embeddings. However, some constructions, even if correctly simulated by our system, are not recommended. For instance, if shear springs are used in triangular or tetrahedral meshes, they only provide the element with more stretching so should be omitted. Furthermore, bending springs are, to our knowledge, rarely

used in 3D (in practice, they control dihedral angles of volumes which is rather redundant with stretching springs).

4.3 Continuous Mechanics Approaches

In this section, the modeling of continuous mechanics is discussed. The same strategy already done with mass/spring system is applied.

4.3.1 Linear FEM

FEM use a tessellation of an object into a finite number of elements. In the following, tetrahedra are used. Each tetrahedron is characterized by a 12x12 stiffness matrix **K** (see (O'Brien and Hodgins, 1999)).

Matrix **K** is computed in a preliminary phase for each tetrahedron and is embedded into volume orbits $\langle 0,1,2 \rangle$. Nevertheless, this matrix depends upon a given order of vertices (positive orientation of the tetrahedron's volume) that must be stored in the structure. Here, we propose to tag only one dart as "fist-dart" for each volume. As shown in Figure 6, the vertex that owns this dart in its orbit is vertex 1. From this dart and following the adjacency relations of the G-maps in a given order, the number of other vertices is recovered.

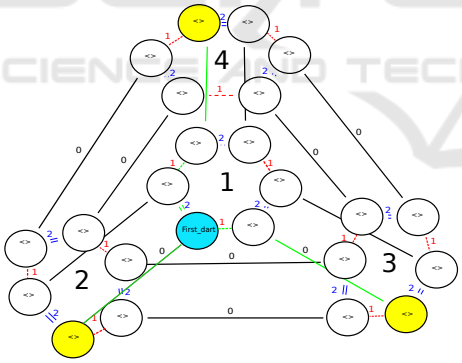


Figure 6: Numbering of vertices.

At each time step and for each tetrahedron, forces must be computed as:

$$\mathbf{f} = -\mathbf{K}\mathbf{u} \tag{3}$$

where **f** includes the forces of all vertices of the tetrahedron, and **u** represents their displacements. Forces computed are stored in the corresponding "corner" of the tetrahedron (orbit $\langle 1,2 \rangle$).

For hexahedra, the approach remains the same, only the information to embed changes. However, modeling with FEM using linear elasticity can simulate only small displacements. To allow rotation

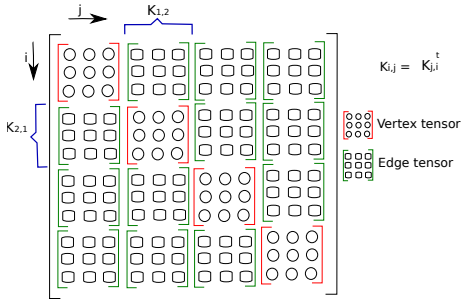


Figure 7: Decomposition of a stiffness matrix.

of objects, the Co-rotational method has been implemented using a QR decomposition (Nesme et al., 2005). The embeddings remain the same, only force computations slightly differ. This method was also used to implement 2D FEM for triangular meshes (Etzmuss et al., 2003).

4.3.2 Linear Mass/Tensor

In this section, it is shown that our model is well adapted to the mass/tensor approach as proposed by (Cotin et al., 2000). This method considers the 12x12 stiffness matrix computed using linear FEM, but decomposes it into sixteen 3x3 matrices $\mathbf{K_{ij}}$. Such a decomposition is shown in Figure 7. Diagonal matrices (red) show the influence of the displacement of a vertex on itself. The other matrices (green) show the influence of any vertex of a tetrahedron on the others.

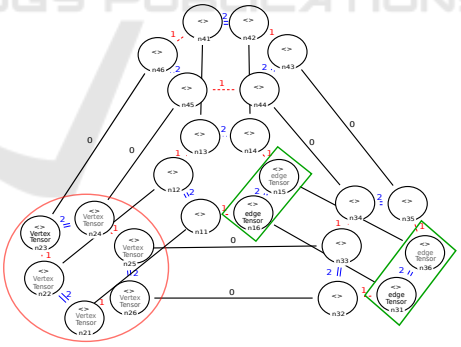


Figure 8: Embeddings for stiffness tensors.

Embeddings are shown in Figure 8, diagonal matrices $\mathbf{K_{ii}}$ can be stored in vertex i, for the current volume (orbits $\langle 1,2 \rangle$). The other matrices $\mathbf{K_{ij}}$ cope with two nodes that are bound by an edge in the volume ($\langle 0,2 \rangle$). However, an edge corresponds both to the influence of vertex i over j and reciprocally, but not using the same matrix. As a consequence, any $\mathbf{K_{ij}}$ is stored on orbit $\langle 2 \rangle$ of vertex i and $\mathbf{K_{ji}}$ on orbit $\langle 2 \rangle$ of vertex j. Note that numbering of vertices is no longer required. On each vertex ($\langle 1,2,3 \rangle$), the sum of all the tensors stored in sub-orbits ($\langle 1,2 \rangle$) is multiplied by its

displacement and force is stored in orbit $\langle 1,2,3 \rangle$. The same way, on each edge extremity ($\langle 2,3 \rangle$), the tensors stored in sub-orbits $\langle 2 \rangle$ are summed to compute force, thereafter stored in orbit $\langle 2,3 \rangle$.

5 IMPLEMENTATION AND RESULTS

As cited above, to test our model and its capabilities, JERBOA , a tool that allows us to implement our rules using a graphic language has been used. Any additional process (mostly computation of embeddings) is written in Java. JERBOA is mainly a prototyping tool, it is not dedicated to fast computation and does not generate optimized code to implement rules. It is used as a proof of concept. Nevertheless, it must be noted that the time of prototyping using JERBOA is unmatched with the habitual time that programmers spend to program the different model deformations. For example, only two days have been spent to add and test the specific rules to extend 2D mass/spring model to 3D, and this time mainly focused on the construction of the simulated object and not the design of interactions (rules remained actually the same). The same design speed up has been observed when generalizing the rectangular meshes to triangles or mixed elements.

For each interaction, one rule is defined The match pattern includes the place where the interaction parameters are embedded and the sub-graph that gives the application vertices. Some conditions can be added: for instance, rules for linear FEM or co-rotational must be applied after having identified the dart tagged as "first-dart". The right part of the rule is the same pattern as on the left, but shows which forces have to be computed. Note that, compared to 2D, 3D rules require to take into account 3-links. So it is advisable, when useful, to write rules directly in 3D, since they can be applied to 2D objects as well. Only rules that explicitly require 3-link in the match pattern are 3D specific. Here, one can understand why the sub-graph must be chosen as general as possible, since it determines the place where the match pattern can be applied. For instance, rules that filter a quad automatically put triangles apart Some rules are also written to build the object or to define/initialize interactions (for instance, a rule aims at computing stiffness matrix for FEM models).

Figure 9 shows an example of a rule to implement stretching spring. In the left part, an edge (that represents the spring) and their two extremities `n0` and `n1` (where the force must be applied) were matched. Here, we can see that a 0-link appears explicitly to

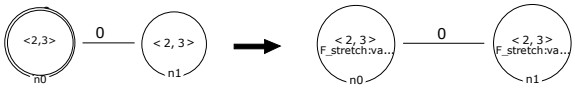


Figure 9: Rule to compute a stretching springs forces.

bring out the two extremities of the edge, and the rest of the edge orbit is expressed in the nodes. The right part expresses that the force computed and stored in the orbit $\langle 2,3 \rangle$ of the vertex `n0` and its opposite have to be stored in the orbit $\langle 2,3 \rangle$ of the vertex `n1`.

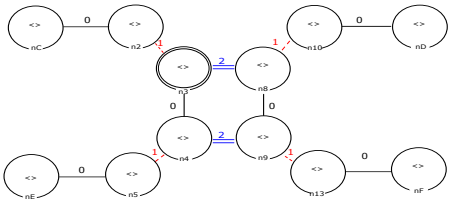


Figure 10: Matched pattern to compute angular springs forces.

Another example that implements angular bending springs is represented in Figure 10. Here, we present only the matched pattern (left part of the rule). The edge which supports the spring constants can be recognized by the four darts `n3`, `n4`, `n8` and `n9`. The sub-graph that allows one to find the vertices that undergo the corresponding forces is also represented (darts `nc`, `nD`, `nE` and `nF`). In the right part of the rule (not presented), the angular spring's forces are computed for all concerned vertices and stored in the correspond darts. The opposite of every force is applied to the dart which is 0-linked (`n2`, `n10`, `n5` and `n13`). Figures 11 and 12 shows some results of our prototyping using mass/spring system and Figure 13, some results using FEM.

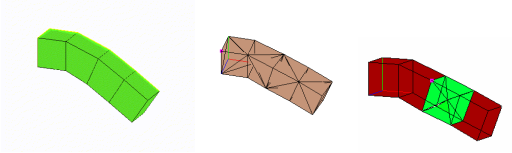
Finally as described in the simulation loop, rules are written to (1) collect all forces distributed in each vertex sub-orbit, (2) find the acceleration, the new velocity and the new position of each vertex. Symplectic Euler has been used to integrate the equations of motion. Combinations of rules can be written if more complex schemes are used.

6 CONCLUSION AND FUTURE WORK

The framework presented in this article aims at providing the existing physical models used in computer graphics with a topological model. G-maps have been chosen since they are more atomic and provide more places to embed information than other models. All mechanical data are stored directly in the G-map, and

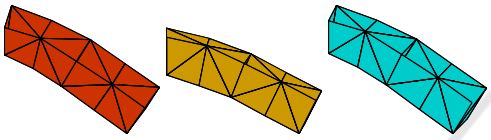
(a) Rectangular (b) Triangular (c) Mixed

Figure 11: Prototypes of mass/spring models in 2D.



(a) Hexahedral (b) Tetrahedral (c) Mixed

Figure 12: Prototyping of mass/spring models in 3D.



(a) FEM linear (b) Co-rotational (c) Mass/tensor

Figure 13: Prototyping of FEM models

the simulation uses neighboring relationships advantageously.

Our model is completely general and generic. First, it is independent on the dimension (2D,3D). Second, it allows discrete or continuous interactions, using the same general modeling process. Third, based on sub-graph matching, our framework copes with different types of elements and allows mixed meshes.

Finally, a rule-based language has been used for prototyping and building our simulation models. It brings a graphical description of interactions and all the necessary walks through the structure.

In future work, we want to test and enhance our model with topological modifications, including mesh adaptation or refinement. Last, considering that a long-term evolution of JERBOA is to generate an optimized code implementing (in various languages including C++) the rules that have been defined, we hope to design a generator of physical models where the user would only detail which forces the model should undergo.

ACKNOWLEDGEMENTS

This work has been partially funded by the European Erasmus Mundus - Al Idrisi II scholarship program. It also received fundings from the MIRES federation.

Special thanks to Daniel Meneveaux for proofreading and comments on this paper.

REFERENCES

Belhaouari, H., Arnould, A., LeGall, P., and Bellet, T. (2014). Jerboa: A graph transformation library for topology-based geometric modeling. In *ICGT*, pages 269–284.

Cotin, S., Delingette, H., and Ayache, N. (2000). A hybrid elastic model allowing realtime cutting, deformation and force feedback for surgery training and simulation. *Visual Computer*, pages 437–452.

Damiand, G. and Lienhardt, P. (2014). *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. A K Peter/CRC Press.

Etzmuss, O., Keckeisen, M., and Strasser, W. (2003). A fast finite element solution for cloth modelling. In *Pacific Conference on Computer Graphics and Applications*, pages 244–251.

Fléchon, E., Zara, F., Damiand, G., and Jaillet, F. (2013). A generic topological framework for physical simulation. In *WSCG*, pages 104–113.

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

Golec, K., Coquet, M., Zara, F., and Damiand, G. (2015). Improvement of a topological-physical model to manage different physical simulation. In *WSCG*, pages 25–34.

Grinspun, E., Hirani, A. N., Desbrun, M., and Schroder, P. (2003). Discrete shells. In *Symposium on Computer Animation*, pages 62–67.

Kjolstad, F., Kamil, S., Kelley, J. R., Levin, D. I. W., Shinjiro Sueda, D. C., Vouga, E., Kaufman, D. M., Kanwar, G., Matusik, W., and Amarasinghe, S. P. (2016). Simit: A language for physical simulations. *ACM Transactions on Graphics*, 35.

Meseure, P., Darles, E., and Skapin, X. (2010). Topology based physical simulation. In *VRIPHYS*, pages 1–10.

Nesme, M., Payan, Y., and Faure, F. (2005). Efficient, physically plausible finite elements. In *Eurographics*, pages 77–80.

O'Brien, J. F. and Hodgins, J. (1999). Graphical modeling and animation of brittle fracture. In *SIGGRAPH*, pages 137–146.

Provot, X. (1995). Deformation constraints in a masss/pring model to describe rigid cloth behaviour. In *Graphics Interface*, pages 147–154.

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. (1987). Elastically deformable models. In *SIGGRAPH*, volume 21, pages 205–214.