Enabling Cuts on Multiresolution Representation

F. Ganovelli, P. Cignoni, C Montani
I.E.I. - C.N.R., Pisa Italy
{*ganovell* | *cignoni* | *montani*}*@iei.pi.cnr.it*

Roberto Scopigno
CNUCE -C.N.R., Pisa Italy
roberto.scopigno@cnuce.cnr.it

Abstract

Multiresolution representations are widely used in many visualization contexts and applications. The adoption of a multiresolution approach provides an optimal management of a data representation, using at each instant of time the level of detail more adequate for the given action or task to be performed. Recently, multiresolution has been introduced also in the interactive physically based simulation of deformable objects (e.g. in virtual surgery applications). In this applications the processing resources available are often insufficient and pose a critical constraint. The adoption of multiresolution allows to improve the accuracy of the simulation in the proximity of the action focus, while maintaining computations under a given bound.
In this particular context, the user should be able to perform cuts onto the object. The problem is that most multiresolution models need a preprocessing phase, in which the data structure is constructed. Such construction strictly depends on the topology of the object, which is supposed to be invariable.
We propose a new approach for the dynamic topological modification of a multiresolution model, which allows easy update of the multiresolution data structure (based on the Multiresolution Triangulation framework) and efficient decomposition of the cells intersected by the cut. With respect to previous methods, our solution supports a much lower degree of fragmentation of the decomposition and very short processing times, due to the design of a LUT-based split solution.

1. Introduction

One of the requirements of a surgery training and simulation software is that the surgeon has to be enabled to cut the represented tissue. This requirement gives rise to the problem of how the cuts change the topology of the mesh. A cut is in general defined by the surface swept by the movement of the scalpel during two consecutive time steps. For a sufficiently small time step, such a surface can be approximated by a portion of a plane. In the case of objects represented by tetrahedral meshes, a trivial solution is to delete the edges intersected by the surface. However, in this manner, we perform an approximated topology change and so we need tetrahedral complexes very dense in order to be sure that such an operation does not introduce a big hole in the mesh. A more sophisticated solution in presented in [1]. This technique replaces each tetrahedron intersecting the surface of the cut with a set of tetrahedra having no proper intersections with it and covering the same space of the original tetrahedron. This technique allows the use of low resolution meshes, because they will result refined by the update, according with the cut. We adopt this kind of approach by providing and efficient implementation which also takes into account possible multiresolution representation of the deformable object.

The paper is organized as follows. An efficient way to implement a multiresolution representation, the MT [9], is briefly showed in Section 3. In Section 4 we show how the MT can be dynamically updated after a modification of the mesh topology induced by a cut. Then we describe, in Section 5, how to replace a cut tetrahedron with a set of tetrahedra not having proper intersections with the plane of the cut in a more efficient way than that proposed in [1]. Finally, results are discussed in Section 6 and concluding remarks are given in Section 7.

2. Previous work

A recent survey on deformable objects modeling can be found in [14]. In this paper, we limit ourselves to briefly describe the solutions with respect to their suitability to the cuts. Some pioneering works employed a continuum representation of the object [21, 20] and the theory of elasticity for modeling the material. Another approach based on a continuum description is the so called Free Form Deformation (FFD) [19]. The idea is that the object is defined as a set of splines whose control points are initially (i.e. when the objects is in its rest state) distributed on a regular grid.

A displacement of the control points implies a change of the shape [3, 4, 10, 12]; In a more recent approach, FFD models adopt physical based behaviour constraining displacement of control points with an energy function [15]. Several authors proposed the use of the Finite Element Method (FEM): the object is partitioned into finite elements joined at discrete points and a function, which solves equilibrium problem locally to each element is found. FEM is, in general, the more accurate method and it was applied to model human tissue for craniofacial surgery planning [11, 18]. Recent works [2, 5] employ FEM also for interactive simulations.

All of the approaches cited above can hardly allow cuts together with interactive frame rate because they require a preprocessing phase which depends on the topology of the object.

Most of the methods proposed in the last years in the computer graphics community are based on particle systems [17]. A particle system is a set of elements (the particles) and a set of relationship between the elements. Each particle is basically described by its position, mass and velocity; a relationship is a description of the interaction between the particles involved. A typical, widely employed, particle system is the Mass Spring System. In this case, the particles are mass points with no dimension and the relations are springs connecting pairs of mass points. The approaches based on particles present two main advantages: they are generally very easy to implement and they do not require initialization. On the other hand, they are not much faster than the other methods, and the computational time strongly depends on the number of particles. For this reason, a current trend in this field is to introduce multiresolution representation to use selectively more particles where an higher accuracy is required, e.g. in proximity of the action focus [8, 7]. Up to now these approaches did not allow cutting. This is because multiresolution is always implemented by static data structures (for example octrees, where each node defines a representation coarser than those of its children). Hence, when a cut is performed its effect affects not only the mesh but also the data structure. By our knowledge, no multiresolution framework provides dynamic updates when the topology of the object changes.

3. The Multiresolution Triangulation

The Multiresolution Triangulation (MT) is a general framework introduced in [6, 16] to manage triangle meshes. Thanks to its generality, MT can be naturally extended to tetrahedral mesh. In the following, we introduce the MT in an intuitive way (see Figure 1) and we explain how to build it during a simplification of a mesh Ω_0 (a more formal treatment can be found in [6]).

Given an incremental simplification process, mesh simplification proceeds through a number of atomic local simplification actions that reduce the mesh size. The main idea of the MT is that we may, with some restrictions, apply back all these mesh updates with an order different from the original simplification sequence. In this way we can decide where we need more or less detail after the simplification process. Obviously, there are some dependencies between local modifications operating on the same area. The MT model codifies all these dependencies as a partial order or, equivalently, with a DAG.

This DAG has a root (a node with no incoming arcs) and a drain (a node with no leaving arcs). The nodes of the DAG are called *fragments*, where a fragment corresponds to a set of tetrahedra created during a simplification atomic action. The root fragment corresponds to the whole initial high resolution mesh Ω_0 ($Root = \Omega_0$).

At the simplification step i, a set of tetrahedra $F_i \in \Omega_{i-1}$ is replaced by Σ_i (note that the tetrahedra F_i can belong to several fragments, added to the DAG in previous steps): we add to the DAG the fragment Σ_i and a set of arcs $\{(\Sigma_j, \Sigma_i, F_i \bigcap \Sigma_j) \mid F_i \bigcap \Sigma_j \neq \emptyset\}$.

In other words, we add to Σ_i an arc incoming from any fragment Σ_j having at least one tetrahedron that has been replaced in the i^{th} simplification step, and the set of labels of such arc is the set of the replaced tetrahedra in Σ_j.

If k is the last simplification step, than the tetrahedra in Ω_k will never be replaced. To complete the MT, we add a node Σ_{k+1}, named *drain*, that ideally corresponds to a simplification step which replaces all the tetrahedra of Ω_k, and all the corresponding arcs as explained above.

The MT allows the extraction of non-uniform Level Of Detail either on the whole mesh domain or on a specified region. We do not describe in detail the extraction algorithms (see [6]): it is enough here to know that any constant or variable resolution mesh extracted from the MT is defined by a *frontier* on the DAG, which is a set of arcs containing exactly one arc for each path from the root to the drain. The corresponding representation is given by the union of all the tetrahedra labeling the arcs of the frontier.

3.1. Error

Inherent to any multiresolution model there is a *error* concept that gives us a value of "fineness" of the cells involved. Usually the cells of the coarser (finest) representation are the ones with the greatest (smallest) error. The definition of error typically depends on the application for which the multiresolution model is used, but in general the error increases with the dimension of the cell (any characteristic of a domain is better expressed with a great number of small cells rather than with a small number of big cells).

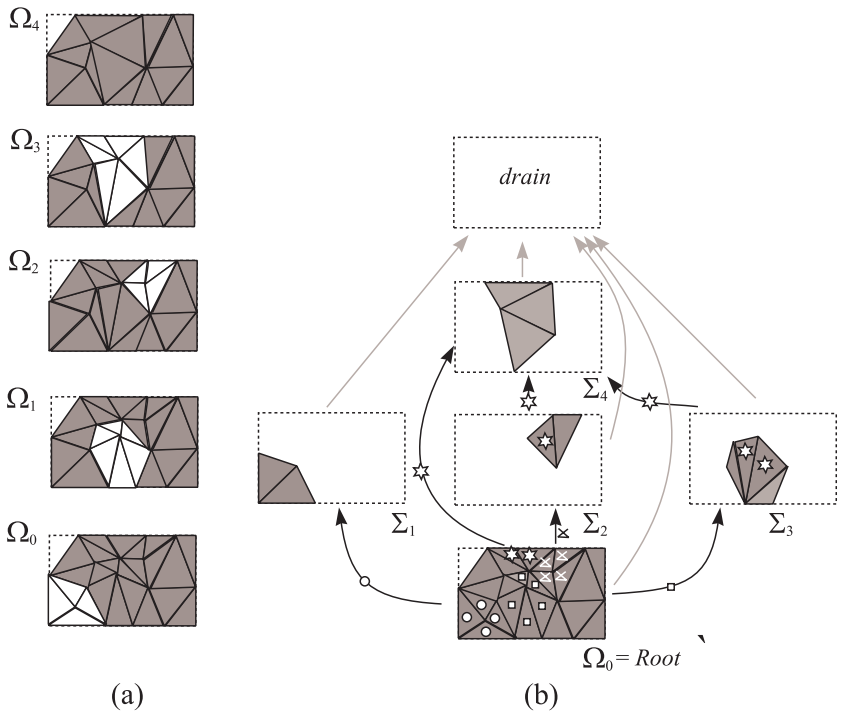

Figure 1. MT construction on a 2D example (a) four decimation steps (b) The relative DAG

We have found that, for the purpose of simulation through mass-spring models, we can consider the error of a tetrahedron proportional to its volume and/or the radius of the smallest sphere containing it.

4. Cutting a Mesh

We have seen that the MT does not allow topological modifications to its structure. In this section we define a cutting operator that incises an MT, with a cutting shape defined as a simple planar convex polygon. In other words we want the cutting polygon modifies the MT in a way that any representation extracted from it has no proper intersections with the given cutting shape. Recalling the "error" definition of Section 3.1, we slightly relax the previous definition allowing to the cutting shape to be approximated inside the MT with an error compatible with the one of the various fragments. In other words we meant that a very small cut has no effect on the coarser representation, while it is represented almost exactly only in the finest resolution.

The input to our algorithm consists of an MT and a planar polygon C which represents the current cut through the mesh. A tetrahedron is cut iff it has a proper intersection with C.
When a cut C is defined, we apply the following steps:

1. Search in Γ for all the cut tetrahedra
2. Replace each cut tetrahedron with a set of tetrahedra having no proper intersections with C
3. Update the DAG

Step 1 can be executed efficiently by exploiting the point location strategy available for MT model [13] and Step 2 will be discussed in detail in the following section.
The third step needs some more explanations. If a fragment Σ_i is divided into two distinct parts by the cut, we replace it with two new fragments $\hat{\Sigma}_i^1$ and $\hat{\Sigma}_i^2$, and the incoming and the outcoming arcs of Σ_i have to be replaced or updated. To achieve a fast update of the DAG, we propose a technique based on local (to the fragment) updates. These updates can be processed in any order (but non in parallel). The technique guarantees that, when the fragments affected by the cut have been processed, the MT (or the two MTs, if the cut divides the mesh) is still correct.
We use $R(t)$ to indicate the tetrahedra originated by the cut of cell t: if t is not cut $R(t) = \{t\}$. In the following we describe the operations for updating, incoming and outcoming arcs, respectively.
To simplify notation in the algorithm, we expand the arc $(\Sigma_k, \Sigma_i, F_i)$ relating the tetrahedra $F_i \subset \Sigma_k$ that were substituted at step i of the simplification process with the ones of the fragment Σ_i into the set of arcs $\{(t_h, \Sigma_i),\ h = 1, \ldots, \|F_i\|\}$, i.e. an arc for each tetrahedron in F_i. Each

fragment has a flag, that we call $separated$. It indicates if the fragment has been already processed and decomposed in two fragments. So $\Sigma_i.separated == true$ indicates that there are two fragments $\hat{\Sigma}_i^1$ and $\hat{\Sigma}_i^2$ originated by the splitting of Σ_i and that it will not be in the new MT.

For each fragment we apply the following procedure:

UpdateProcessFragment(Σ_i, C)
  if IsSplit(Σ_i,C)
    Decompose(Σ_i)
    $\Sigma_i.separated = true$
    UpdateInComingArcs(Σ_i)
  UpdateOutcomingArcs(Σ_i)

IsSplit returns *true* iff the fragment is split by the cut C, i.e. iff the contour of C intersects no tetrahedron of Σ_i.
The **Decompose** procedure effectively split the fragment Σ_i into two different fragments $\hat{\Sigma}_i^1$ and $\hat{\Sigma}_i^2$ (see Fig. 2), and the two procedure UpdateIncomingArcs and UpdateOutcomingArcs are explained in the following.

UpdateIncomingArcs(Σ_i)
  for each (t, Σ_i)
    if $t \bigcap C = \emptyset$
      if $t \bigcap \hat{\Sigma}_i^1 = t$
        replace (t, Σ_i) with $(t, \hat{\Sigma}_i^1)$
      else // $t \bigcap \hat{\Sigma}_i^2 = t$
        replace (t, Σ_i) with $(t, \hat{\Sigma}_i^2)$

UpdateOutcomingArcs(Σ_i)
  for each (t, Σ_k) // $t \in \Sigma_i$
    if $\Sigma_k.separated$
      for each $t' \in R(t)$
        if ($t' \bigcap \hat{\Sigma}_k^1 = t'$)
          add $(t', \hat{\Sigma}_k^1)$
        else // $t' \bigcap \hat{\Sigma}_k^2 = t'$
          add $(t', \hat{\Sigma}_k^2)$
    else
      replace (t, Σ_k) with $\{(t', \Sigma_k) : t' \in R(t)\}$

Note that, if the fragment to which t belongs has not been processed, it is possible that $t \bigcap C$ i.e. it stand across the cut C. In that case the updating of the cut (t, Σ_i) will be done when processing such fragment.

Figure 2 shows an example of this step.

Suppose to have the relations depicted in Figure 2.a with the heavy line representing the cut. If the fragment Σ_0 is processed before Σ_1, the arcs $(5, \Sigma_1)$ and $(6, \Sigma_1)$ will be replaced with the arcs $(18[19, 20], \Sigma_1)$ and $(21[22, 23], \Sigma_1)$. When processing the fragment Σ_1, the arcs $(18[19, 20], \Sigma_1)$ become $(18[19, 20], \hat{\Sigma}_1^1)$ and the arcs $(21[22, 23], \Sigma_1)$ become $(21[22, 23], \hat{\Sigma}_1^2)$.

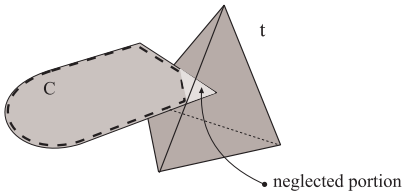

Figure 3. The portion of plane C intersects the tetrahedron t without to touch its edges. The approximating cutting surface is the one dashed.

Viceversa, if the fragment Σ_1 is processed before, the arcs $(12[13], \Sigma_1)$ become $(12[13], \hat{\Sigma}_1^1)$ and the arcs $(5[6], \Sigma_1)$ do not change. When processing the fragment Σ_0, the arcs $(5, \Sigma_1)$ and $(6, \Sigma_2)$ they will become respectively $(18[19, 20], \hat{\Sigma_1})_1^1$ and $(21[22, 23], \hat{\Sigma}_1^2)$.

Hence, in both cases the final situation is the one depicted in Figure 2.b.

5. Cutting tetrahedral cells

So far, we have assumed to be able, given a portion of the plane C, to replace a tetrahedron t with a set of tetrahedra $R(t)$ so that the union of the space occupied by tetrahedra in $R(t)$ is equal to the space occupied by t and no tetrahedra in $R(t)$ has a proper intersection with the half plane C. We describe in detail the way we perform the atomic split efficiently, both in time and size of the output.

5.1. Approximating a cutting shape

When C is defined, the replacing operation of a tetrahedron t depends on the topology of its intersection with C. We want to have a finite number of possible topologies, so we consider only the ones defined by the intersections of the edges of t with C. In other words, if C intersects one or more faces of t without intersect any edges of t, we disregard such intersection (see Figure 3).
The error introduced by this approximation depends on the shape and dimension of C relatively to the mesh. If C is a segment, the worst case, it could intersect a great number of tetrahedra producing no effects. In general, the approximation error resides entirely on the contour of C.

A solution for cell cutting was proposed in [1]. The authors observe that, if the but on a tetrahedron is defined by the set of intersections of the swept surface with its edges, there are only 5 possible configurations of a cut. They use a Look Up Table (LUT) to encode and select, at running time, the different configurations: each entry in the LUT

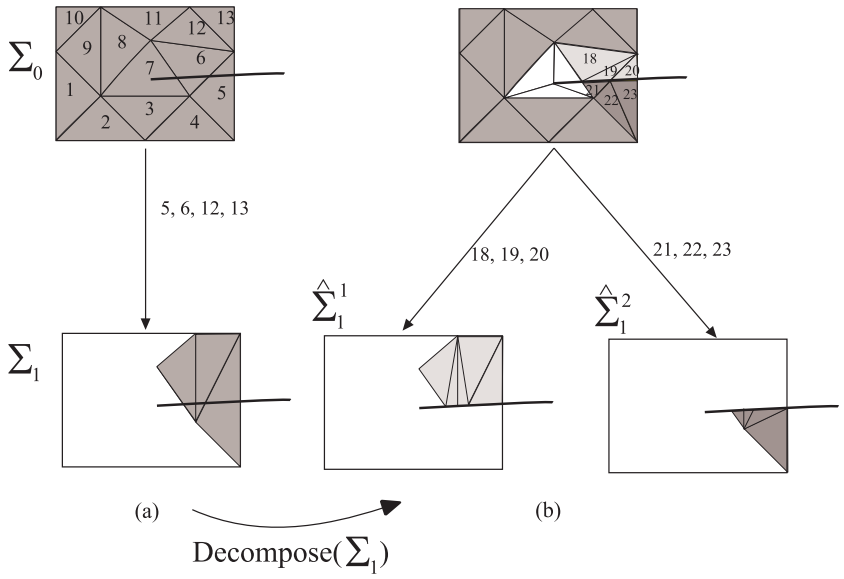

Figure 2. An example of the algorithm for dinamic update of the DAG

stores the set of *faces* to add to the surface to match with the cut performed. On the other hand, for defining the set of tetrahedra which are replaced to the cut one, they adopt a standard 1:17 splitting scheme (computed in a preprocessing phase, and shown in Figure 4) for any cut occurrence. This splitting pattern covers all the possible configurations arising from any partial or complete cut of a tetrahedron, with the restriction previously introduced. On the other hand, because they do not differentiate between different cutting instances, the number of tetrahedral cells generated is often not the optimal one (since each tetrahedron affected by the cut is replaced with seventeen new cells). In fact, the authors experimented a great number of new tetrahedra in the proximity of a cut. This mesh refinement effect becomes even more critical when multiple cuts are operated on the same portion of the represented tissue.

We improve this technique, using the LUT also for storing a tetrahedra decomposition scheme for each configuration. Each entry stores a set of quadruples, each one defining a single tetrahedron to be produced in output. Each quadruple identifies the vertices of each split tetrahedron (chosen from the vertices of the original cell and the possible intersection points between the cut and either the cell edges or the cell face); the encoding adopted is illustrated in Figure 6. We show the five possible configurations and the set of new tetrahedra produced in Figure 5; obviously,the cases, produced by rotating the base cell, are also possible.

We encode two topologically different intersection points (Figure 6) for each original cell edge. Intersection points on the edges are duplicated because the goal of a cut is to allow the separation of the parts, and therefore we need different cell nodes for each border of the cut. Conversely,

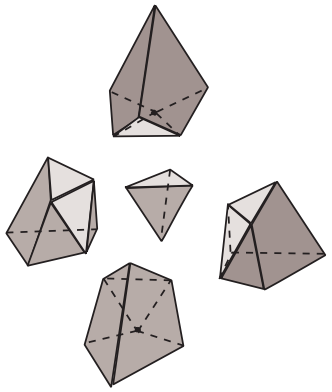

Figure 4. Pre-defined 1:17 splitting scheme, applied to any cut instance. Each polyhedral cell is then decomposed into tetrahedra.

points on the faces do not need to be duplicated because they represent the boundary profile of the cut (case C,D and E in Figure 5).
For example, the first row of the LUT is:

A	0,4,6,8	5,1,7,9	1,3,7,9	1,3,2,7

and it encodes the four tetrahedra associated with a type A cut (three edges intersected by the cut, the tetrahedral cell split into two disconnected parts, see Figure 5).

Our approach uses 4 tetrahedra for the case **A**, 6 for the case **B**, 6 for the case **C**, 8 for the case **D** and 9 for the case **E**.
In the results section (see Table 1) we report an empirical

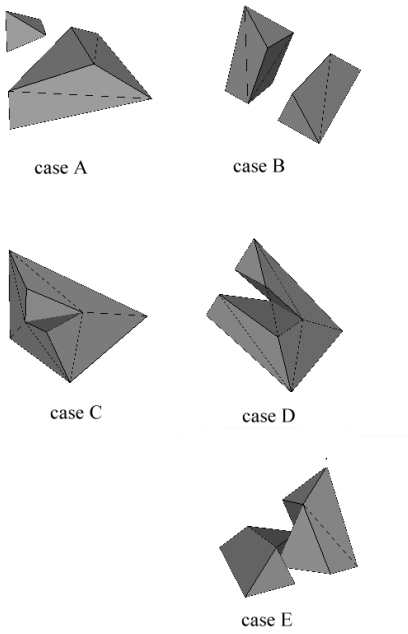

Figure 5. The five possible configurations of a partial or complete cut of a tetrahedral cell.

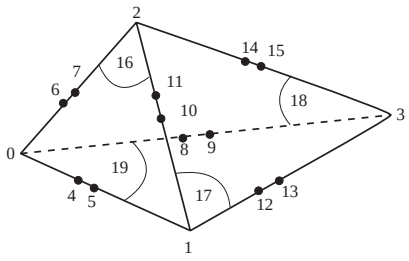

Figure 6. Encoding rule used to define the possible split cell vertices in the LUT.

Sp F	Sp T	1:17	LUT	t Tet	t DAG	tot
205	2,520	42,840	11,612	277	117	394
36	523	8,891	2,394	63	156	312
148	1,813	30,821	8,300	250	109	359
245	3,122	53,0474	14,316	344	141	485
528	6,552	110,874	29,918	531	344	875

Table 1. Comparison between the tetrahedra set generated by the pre-split scheme and by our LUT-based split rule. The original MT is composed by 2,896 vertices, 37,062 tetrahedra and 1,542 nodes (fragments).

comparison between the use of our LUT-based split rule and the 1:17 pre-split scheme proposed in [1]. The two approaches are compared with respect to the number of new tetrahedra created.

The lower mesh refinement ensured by the LUT-based split rule is obtained with a slight time overhead. For each cell to be split we only need to classify its vertices with respect to the split plane (four comparisons), and then to compute the location of all the new vertices encoded in the corresponding LUT entry. An empirical evaluation of the overall running time is also proposed in Section 6.

6. Results

Figure 7 shows a run of the algorithm on a mechanical piece (Fig.7.a). The first step is to place the tool used to define interactively the cut: we use a disk in the example (Fig.7.b). Figure 7.c shows the two pieces arising from the split of the original piece with the new tetrahedra (rendered darker). Two independent MTs have been produced to represent the resulting two pieces. In the last image (Fig.7.d) the same pieces are shown from a different view point and are represented with two different levels of detail. In Table 1, the first row of contains the data related to the snapshots of Fig.7.

Each row of the table contains, respectively: the number of fragments and the number of tetrahedra involved in the cut, the number of tetrahedra created using the $1:17$ splitting scheme and using our solution LUT based, the times for creating the tetrahedra, updating the DAG and the total time (the time is expressed in milliseconds).

To introduce our solution in a virtual surgery application, two considerations on the computational efficiency have to be done:

- the typical manner to simulate the cut operated with a virtual scalpel is to detect its position on consecutive time steps and to consider the surface swept (the

scalpel is supposed to be a segment) as the surface cut. The time resulting from our tests are related to a cut that involves many tetrahedra, such that the object is separated into two parts. It would correspond to the not very realistic case of trespassing the whole object with the scalpel in two consecutive time steps. Note that the running times in Table 1 are linearly proportional to the number of *tetrahedra* involved in the cut and that the time to update the DAG is proportional to the number of *fragments* involved in the cut.

- when a cut is defined, the algorithm updates the mesh for any resolution (i.e. also the tetrahedra which are not currently visualized/used) and the DAG. Even if no criteria about the order in which the updates of the fragments are made, nevertheless the procedure is quite fast. However, we cannot neglect that, in a surgical simulator, other operations have to be done in a time step, i.e. integration of the differential equations governing the system, collision detection and response feedback. A time-critical version of the algorithm can be simply obtained by processing those fragments which are currently visualized first, and postponing the others to successive time steps, having the only restriction that the resolution cannot change before all the fragments have been processed and updated.

7. Conclusion and Future Works

This works has been inspired by the need to perform cuts on meshes representing soft tissues in a mutiresolution framework. Both multiresolution and cuts have been previously introduced in deformable object modeling: unfortunately their employment was mutually exclusive. We have defined an approach to fill this lack in the case of tetrahedral representations; based on the adoption of the MT multiresolution scheme. In spite of the fact that the implementation can be further optimized, the experiments show the efficiency of the solution proposed. Our next step will be to adopt this approach in a prototypal system which implements physical simulation.

Even if simulating a cut is a fundamental feature of a virtual surgery system, the inverse operation as well, the suture, should have to be treated. A lazy idea to define virtual suture is simply to define pairs of landmarks on two different surfaces and then to join them according with such points. This would be a nice tools, but useless, because a realistic suture procedure implies to sew, i.e. to join with a needle and thread, not simply to indicate the pair of points to join. More research is needed on this issue.

We conclude introducing a straightforward application of this work, which is not connected with virtual surgery. The MT scheme allows to perform region interference queries within a given error, i.e. it is possible to choose a region of interest and to perform a query only in such region, reducing the time required to solve the query. Note that, even if the region of interest is always the same in different queries, the whole DAG has to be kept in memory. On very large datasets this can be a problem, due to limited RAM size. With our algorithm, we can easily create a MT which corresponds to the specified region, allowing the user to work only with the amount of data related to the region of interest.

References

[1] D. Bielser, V. Maiwald, and M. Gross. Interactive cuts through 3-dimensional soft tissue. *Computer Graphics Forum (Eurographics'99 Proc.)*, 18(3):C31–C38, Sept. 1999.

[2] M. Bro-Nielsen and S. Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 15(3):C57–C66, C461, Sept. 1996.

[3] S. Coquillart. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. *Computer Graphics (SIGGRAPH '90 Proceedings)*, 24(4):187–196, Aug. 1990.

[4] S. Coquillart and P. Jancene. Animated free-form deformation: An interactive animation technique. *Computer Graphics*, 25(4):23–26, July 1991.

[5] H. D. S. Cotin and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *CAS99 Proceedings*, pages 70–81, May 1999.

[6] L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multiresolution modeling. In R. Klein, W. Straßer, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pages 302–323. Springer-Verlag, 1997.

[7] G. Debunne, M. Desbrun, A. Barr, and M. Cani. Interactive multiresolution animation of deformable models. In N.Magnenat-Thalmann and D.Thalmann, editors, *Eurographics Workshop on Computer Animation and Simulation '99*, pages 133–144. Springer-Verlag, Sept. 1999.

[8] P. C. F. Ganovelli and R. Scopigno. Introducing multiresolution representation in deformable modeling. In J. Zara, editor, *SCCG '99 Conference Proceedings, Budmerice (Slovakia)*, pages 149–158, April 28th-May 1st 1999.

[9] L. D. Floriani, P. Magillo, and E. Puppo. Efficient implementation of multi-triangulations. In *Proceedings IEEE Visualization'98*, pages 43–50. IEEE, 1998.

[10] W. M. Hsu, J. F. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, 26(2):177–184, July 1992.

[11] E. Keeve, S. Girod, P. Pfeifle, and B. Girod. Anatomy-based facial tissue modeling using the finite element method. In *IEEE Visualization '96*. IEEE, Oct. 1996. ISBN 0-89791-864-9.

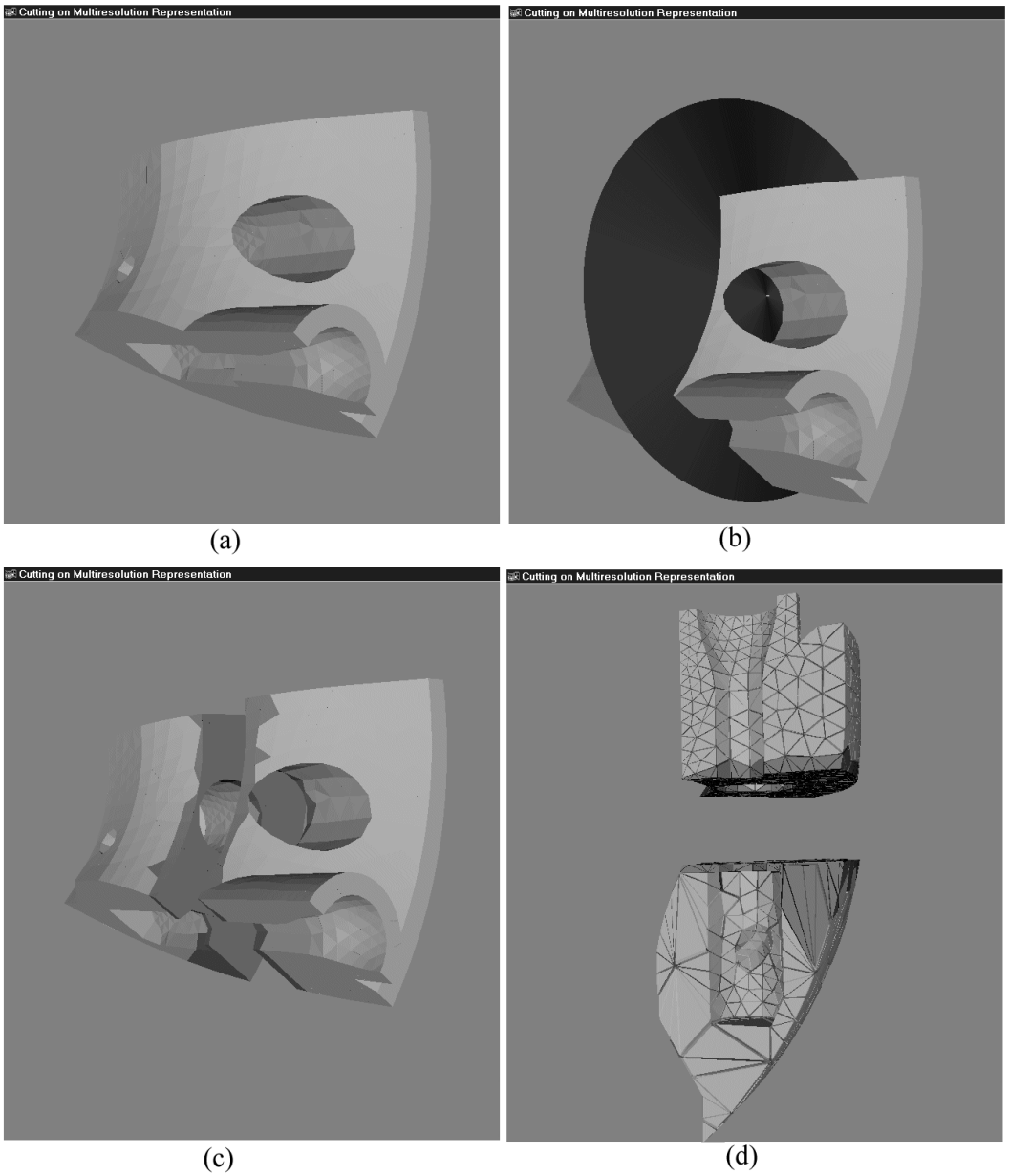

Figure 7. Snapshots of the application window. The MT associated with the piece (a) is split in two independent MTs associated with the two parts in (d).

[12] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. *Computer Graphics (Annual Conference Series)*, 30:181–188, 1996.

[13] P. Magillo. *Spatial Operations on Multiresolution Cell Complexes*. PhD thesis, Università degli Studi di Genova, 1999.

[14] U. M. C. F. A. M.Prati. Modelling and haptic interaction with non-rigid materials. *Computer Graphics Forum (Eurographics'99 Proc.)*, 18(3):1–20, Sept. 1999.

[15] P. F. M. V. Panne and D.Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, 1997.

[16] E. Puppo. Variable resolution terrain surfaces. In *Proceedings Eight Canadian Conference on Computational Geometry, Ottawa, Canada*, pages 202–210, August 12-15 1996.

[17] W. T. Reeves. Particle systems – A technique for modeling a class of fuzzy objects. *Computer Graphics (SIGGRAPH '83 Proceedings)*, 17(3):359–376, July 1983.

[18] S. H. M. Roth, M. H. Gross, S. Turello, and F. R. Carls. A Bernstein-Bèzier based approach to soft tissue simulation. *Computer Graphics Forum*, 17(3):285–302, 1998.

[19] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, Aug. 1986.

[20] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, Dec. 1988.

[21] D. Terzopoulos, J. Platt, A. Barr, and K. Fleisher. Elastically deformable models. *Computer Graphics, June 1987*, 21(4):205–214, 1987.