

Enabling cuts on multiresolution representation

F. Ganovelli, P. Cignoni,
C. Montani, R. Scopigno

Istituto di Scienza e Tecnologie dell'Informazione¹ –
Consiglio Nazionale delle Ricerche, Area della
Ricerca CNR, S. Cataldo, 56100 Pisa, Italy
e-mail: {ganovelli, cignoni, montani}@iei.pi.cnr.it,
r.scopigno@cnuce.cnr.it

Multiresolution representations are widely used in many visualization contexts and applications, since they provide optimal management of the data representation by using at each time instant a level of detail most appropriate for the application requirements. Unfortunately, current solutions do not allow the topology of the object to be changed, and this tends to prevent its adoption in applications where topological changes are needed, such as virtual surgery applications. By extending a known multiresolution model based on simplicial complexes, we develop a new approach which supports dynamic topological modifications of the represented object without greatly increasing the representation complexity.

Key words: Multiresolution – Simplicial complexes – Deformable object modelling

¹ Formerly IEI-CNR and CNUCE-CNR

1 Introduction

A virtual surgery system is a typical application where topological changes have to be applied to the models of the objects represented. As an example we can consider a system for simulating a cholecystectomy: the gall bladder in the system should be modeled as an object which can be cut. The most relevant characteristic of these models is that the object deforms according to the external actions performed by the user and (in general a subset of) the physical laws of the materials involved. This research field is usually termed *deformable object modeling* (DOM). Among the large number of models proposed in the literature, only a few allow cuts on the object (surveys can be found in Cugini et al. 1999; Gibson and Mirtich 1997). The difficulty resides in the fact that the more physically accurate and efficient the model, the more it needs a preprocessing phase which depends on the topology of the object. A straightforward example comes from the models based on the finite element method (FEM). The object is represented by means of a partition (commonly called the “mesh”) of finite elements for which the behaviour in terms of reaction to external actions is pre-computed; the relations among the elements is arranged in a such way that, at run time, the new shape of the object under load is easily obtained by solving a linear system. Any modification to the mesh will invalidate part of the preprocessing phase, which must then be performed again, requiring a computational time incompatible with interactive solutions (although works in this exist: direction Berkley et al. 1999; Debunne et al. 2000; Delingette et al. 1999; Hutchinson et al. 1996; Nienhuys and van der Stappen 2000).

Models using multiresolution representations are another example; their advantage comes from the ability to provide a varying levels of detail for the description of the object in time and space, in order to optimize CPU use. Multiresolution representations can be found in the visualization of terrains (De Floriani et al. 1998), where the ground is represented with a level of detail decreasing with the distance from the user, or in DOM system simulation (Ganovelli et al. 1999; Debunne et al. 1999), where the level of detail is higher in the regions where more accuracy is required. As in the case of FEM, the multiresolution approach requires an off-line preprocessing phase. In this phase the alternative representations of (parts of) the object are created together with the rules for combining them, in order to efficiently provide a timely representation at run time.

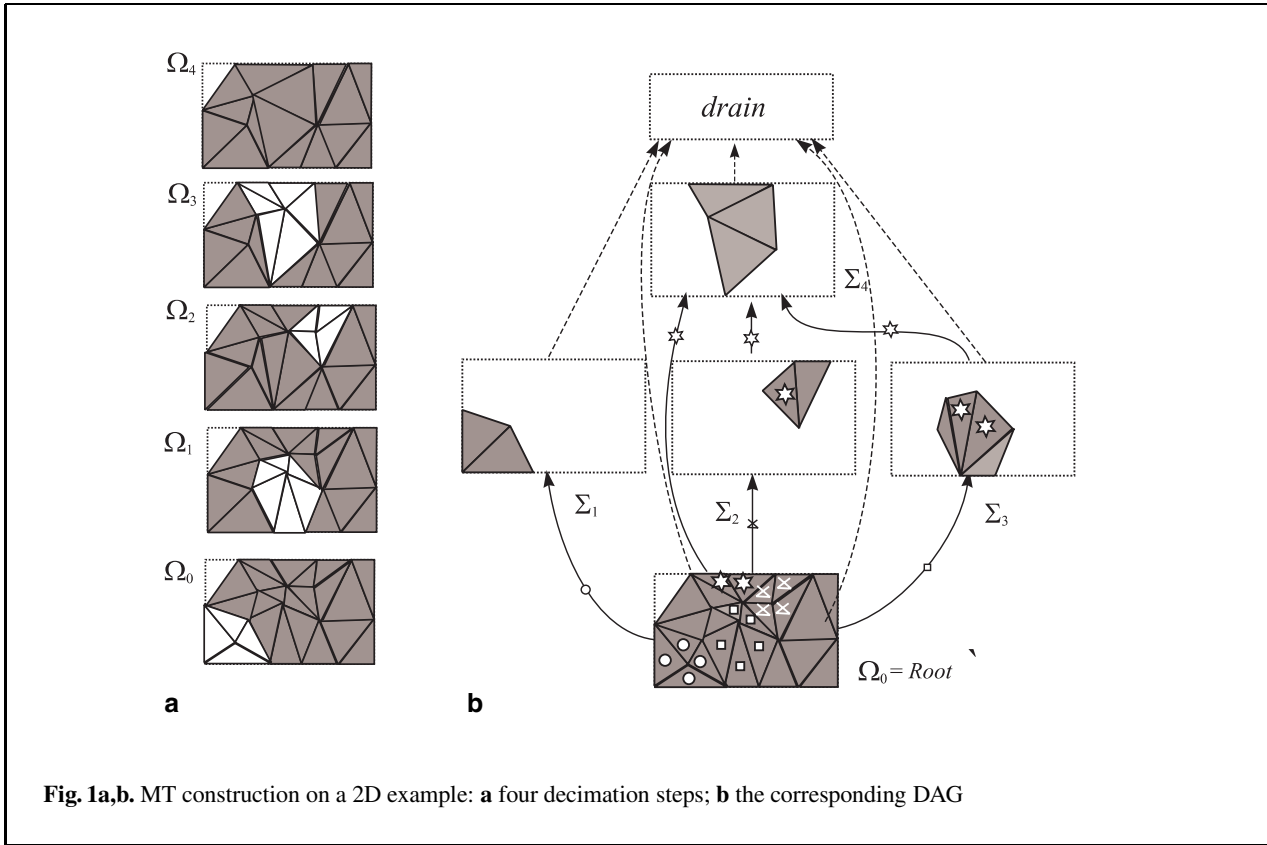


Fig. 1a,b. MT construction on a 2D example: a four decimation steps; b the corresponding DAG

Our goal in this work is to design a multiresolution framework which allows us to perform cuts on the represented object by updating on-the-fly the structures used for the multiresolution representation. In the following, we will work under the hypothesis that a cut is defined as a portion of a plane defined by the user; this is a realistic assumption, since the cut can be considered to be defined by the surface swept by a segment representing a virtual, locally planar scalpel.

The rest of the paper is organized as follows. An efficient way to implement a multiresolution representation, the MT (De Floriani et al. 1998), is briefly described in Sect. 2. In Sect. 3 we show how the MT can be dynamically updated after a modification of the mesh topology induced by a cut. Then, in Sect. 4, we describe how to replace a cut tetrahedron with a set of tetrahedra which do not have proper intersections with the plane of the cut, in a more efficient way than that proposed in (Bielser et al. 1999). Finally, results are discussed in Sect. 6 and concluding remarks are given in Sect. 7.

2 The multiresolution triangulation

The multiresolution triangulation (MT) is a general framework introduced in (De Floriani 1997; Puppo 1996) to manage triangle meshes. Thanks to its generality, MT can be naturally extended to tetrahedral meshes. In the following, we introduce the MT in an intuitive way (Fig. 1) and we explain how the construction of the MT for a generic mesh Ω_0 can be performed during the simplification of the mesh Ω_0 itself. For a more formal treatment we direct the reader to the paper De Floriani et al. (1997).

Given an incremental simplification process, the mesh simplification proceeds through a number of atomic local simplification actions which reduce the mesh size. The main idea of the MT is that it is possible, with some restrictions, to re-apply all these mesh updates in a different order with respect to the original simplification sequence. In this way, it can be dynamically decided where more (or less) detail is needed. Obviously, there are some dependencies among local modifications requiring the same area.

The MT model codifies all these dependencies as a partial order or, equivalently, with a DAG.

The DAG has a root (a node with no incoming arcs) and a drain (a node with no outgoing arcs). The nodes of the DAG are called *fragments*: a fragment corresponds to a set of tetrahedra created during an atomic simplification action. The root fragment corresponds to the entire initial high-resolution mesh Ω_0 .

At simplification step i , if 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), then we add to the DAG the fragment Σ_i and a set of arcs $\{(\Sigma_j, \Sigma_i, F_i \cap \Sigma_j) \mid F_i \cap \Sigma_j \neq \emptyset\}$. In other words, we add to Σ_i an incoming arc 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 tetrahedra replaced in Σ_j .

If k is the last simplification step, then the tetrahedra in Ω_k are not replaced. To complete the MT, we add a node Σ_{k+1} , called the *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 a non-uniform level of detail either on the whole mesh domain or on a specified region. We do not describe in detail the extraction algorithm (De Floriani et al. 1997); we simply emphasize 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.

2.1 Error

Any multiresolution model has to provide an *error* concept that indicates the *approximation accuracy* guaranteed by any cell in the multiresolution representation with respect to the detailed initial model. Usually, the cells of the coarsest (finest) representation are the ones with the greatest (smallest) error. The definition of the 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 (in general, any characteristic of a domain is better represented with a large number of small cells rather than a small number of large cells).

Although we consider, in our experiments, the error of a tetrahedron to be proportional to its volume, the approach is completely independent of the error function associated with the tetrahedra.

3 Cutting a mesh

In the standard definition, the MT does not allow topological modifications to its structure. In this section we define a cutting operator able to cut into an MT, with a cutting shape defined as a simple planar convex polygon. In other words, the cutting polygon should modify the MT in such a way that any representation extracted from it should have no proper intersections with the given cutting shape.

The input of our algorithm consists of an MT and a planar polygon C representing 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 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 in the MT scheme (Magillo 1999); step 2 will be discussed in detail in the following section. The third step needs explanation. If a fragment Σ_i is divided into two distinct parts by the cut, then we replace it with two new fragments $\hat{\Sigma}_i^1$ and $\hat{\Sigma}_i^2$, and the incoming and the outgoing arcs of Σ_i have to be replaced or updated. To achieve rapid update of the DAG, we propose a technique based on local (to the fragment) updates. These updates can be processed in any order (but not in parallel). The technique guarantees that the MT (or the two MTs, if the cut divides the mesh into two halves) is still correct after the fragments affected by the cut have been processed.

We use $R(\sigma)$ to indicate the tetrahedra created by the cut of cell σ : if σ is not cut, then $R(\sigma) = \{\sigma\}$. Below we describe the operations for updating the incoming and outgoing arcs, respectively. To simplify the notation in the algorithm, we expand the arc $(\Sigma_k, \Sigma_i, F_i)$ – associated with the tetrahedra $F_i \subset \Sigma_k$ substituted at step i of the simplification process by those of the fragment Σ_i – into the set

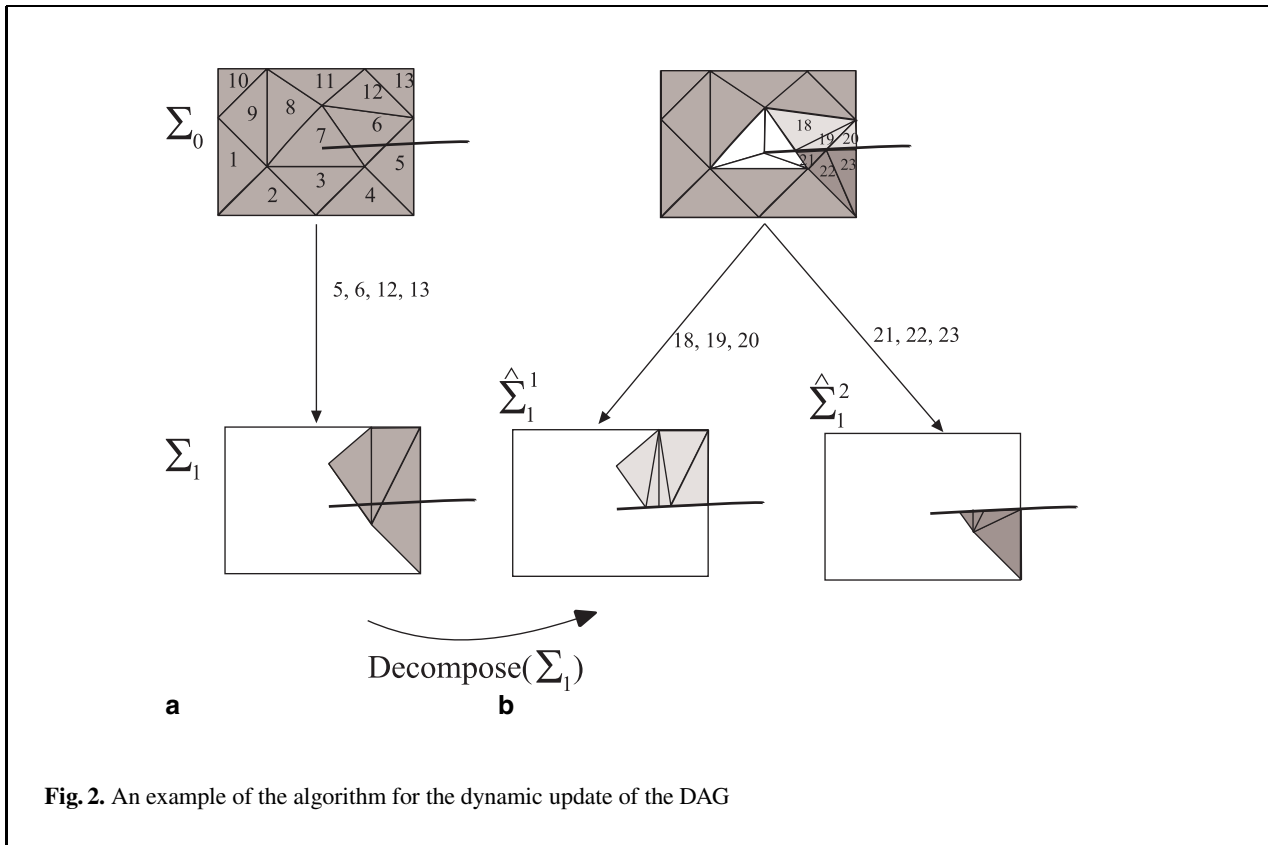


Fig. 2. An example of the algorithm for the dynamic update of the DAG

of arcs $\{(\sigma_h, \Sigma_i), h = 1, \dots, \|F_i\|\}$, i.e. an arc for each tetrahedron in F_i . Each fragment has a *split* flag indicating whether or not the fragment has already been processed and decomposed into two fragments. Thus $\Sigma_i.split == true$ indicates that (a) there are two fragments $\hat{\Sigma}_i^1$ and $\hat{\Sigma}_i^2$ created by the splitting of Σ_i ; and (b) Σ_i will no longer be present in the new MT.

For each fragment the following procedure is applied:

```

UpdateProcessFragment( $\Sigma_i, C$ )
  if IsSplit( $\Sigma_i, C$ )
    Decompose( $\Sigma_i$ )
     $\Sigma_i.split = true$ 
    Update IncomingArcs( $\Sigma_i$ )
    Update OutgoingArcs( $\Sigma_i$ )
  
```

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 splits the fragment Σ_i into two different fragments $\hat{\Sigma}_i^1$ and $\hat{\Sigma}_i^2$ (Fig. 2).

```

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

```

UpdateOutgoingArcs( $\Sigma_i$ )
  for each ( $\sigma, \Sigma_k$ ) //  $\sigma \in \Sigma_i$ 
    if  $\Sigma_k.split$ 
      for each  $\sigma' \in R(\sigma)$ 
        if ( $\sigma' \cap \hat{\Sigma}_k^1 = \sigma'$ )
          add ( $\sigma', \hat{\Sigma}_k^1$ )
        else //  $\sigma' \cap \hat{\Sigma}_k^2 = \sigma'$ 
          add ( $\sigma', \hat{\Sigma}_k^2$ )
        else replace ( $\sigma, \Sigma_k$ ) with  $\{(\sigma', \Sigma_k) : \sigma' \in R(\sigma)\}$ 
  
```

```

UpdateOutgoingArcs( $\Sigma_i$ )
  for each ( $\sigma, \Sigma_k$ ) //  $\sigma \in \Sigma_i$ 
    if  $\Sigma_k.split$ 
      for each  $\sigma' \in R(\sigma)$ 
        if ( $\sigma' \cap \hat{\Sigma}_k^1 = \sigma'$ )
          add ( $\sigma', \hat{\Sigma}_k^1$ )
        else //  $\sigma' \cap \hat{\Sigma}_k^2 = \sigma'$ 
          add ( $\sigma', \hat{\Sigma}_k^2$ )
        else replace ( $\sigma, \Sigma_k$ ) with  $\{(\sigma', \Sigma_k) : \sigma' \in R(\sigma)\}$ 
  
```

```

UpdateOutgoingArcs( $\Sigma_i$ )
  for each ( $\sigma, \Sigma_k$ ) //  $\sigma \in \Sigma_i$ 
    if  $\Sigma_k.split$ 
      for each  $\sigma' \in R(\sigma)$ 
        if ( $\sigma' \cap \hat{\Sigma}_k^1 = \sigma'$ )
          add ( $\sigma', \hat{\Sigma}_k^1$ )
        else //  $\sigma' \cap \hat{\Sigma}_k^2 = \sigma'$ 
          add ( $\sigma', \hat{\Sigma}_k^2$ )
        else replace ( $\sigma, \Sigma_k$ ) with  $\{(\sigma', \Sigma_k) : \sigma' \in R(\sigma)\}$ 
  
```

```

UpdateOutgoingArcs( $\Sigma_i$ )
  for each ( $\sigma, \Sigma_k$ ) //  $\sigma \in \Sigma_i$ 
    if  $\Sigma_k.split$ 
      for each  $\sigma' \in R(\sigma)$ 
        if ( $\sigma' \cap \hat{\Sigma}_k^1 = \sigma'$ )
          add ( $\sigma', \hat{\Sigma}_k^1$ )
        else //  $\sigma' \cap \hat{\Sigma}_k^2 = \sigma'$ 
          add ( $\sigma', \hat{\Sigma}_k^2$ )
        else replace ( $\sigma, \Sigma_k$ ) with  $\{(\sigma', \Sigma_k) : \sigma' \in R(\sigma)\}$ 
  
```

Note that if the fragment to which σ belongs has not been processed, it is possible that σ intersects the cut C . In this case the update of the cut (σ, Σ_i)

will be done when processing this fragment. Figure 2 shows an example of this step. Suppose we have the relations depicted in Fig. 2a with the heavy line representing the cut. If fragment Σ_0 is processed before Σ_1 , then the arcs (5, Σ_1) and (6, Σ_1) will be replaced with the arcs (18[19, 20], Σ_1) and (21[22, 23], Σ_1). When processing fragment Σ_1 , the arcs (18[19, 20], Σ_1) become (18[19, 20], $\hat{\Sigma}_1^1$) and the arcs (21[22, 23], Σ_1) become (21[22, 23], $\hat{\Sigma}_1^2$). On the other hand, if fragment Σ_1 is processed before Σ_0 , then the arcs (12[13], Σ_1) become (12[13], $\hat{\Sigma}_1^1$) and the arcs (5[6], Σ_1) do not change. When processing fragment Σ_0 , the arcs (5, Σ_1) and (6, Σ_2) will become respectively (18[19, 20], $\hat{\Sigma}_1^1$) and (21[22, 23], $\hat{\Sigma}_1^2$). Therefore, in both cases the final situation is that depicted in Fig. 2b.

4 Cutting tetrahedral cells

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

4.1 Approximating a cutting shape

When C is defined, the operation to replace a tetrahedron σ depends on the topology of its intersection with C . We want to have a finite number of possible topologies, so we consider only the cases where the split can be defined by simply looking at the intersections of the edges of σ with C . In other words, we disregard the possibility where C intersects one or more faces of σ without intersecting any edge of σ (Fig. 3). The error introduced by this approximation depends on the shape and dimension of C relative to the mesh. If C is a segment, the worst case, it could intersect many tetrahedra without having any effect. It is easy to see that this approximation error depends entirely on the contour of C .

Under the given assumptions, it has been shown in (Bielser et al. 1999) that there are only five different ways in which a tetrahedron can be intersected by the sweeping surface. These are called *cut configurations*, and are represented in Fig. 4. In the first two configurations, S intersects three or four edges

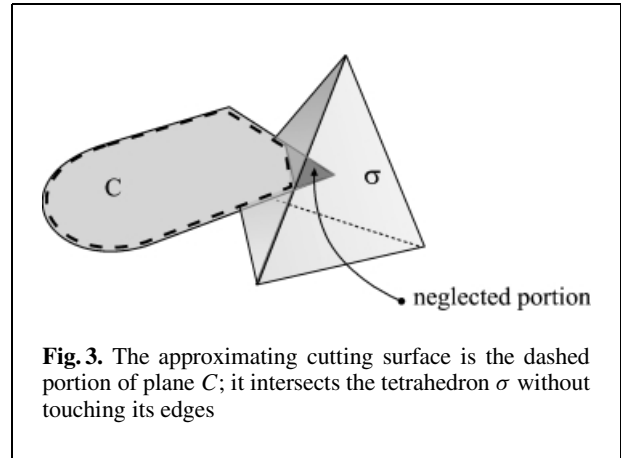


Fig. 3. The approximating cutting surface is the dashed portion of plane C ; it intersects the tetrahedron σ without touching its edges

and splits the tetrahedron into two parts. In the last three cases one, two, or three edges are intersected, and this does not cause a complete split of the tetrahedron. A look up table (LUT) is used to encode and select, at running time, the different configurations: each entry in the LUT stores the set of *faces* to be added to the surface to match the cut performed, while the problem of how to find the set $R(\sigma)$ is solved using a fixed 1 : 17 pre-computed scheme (Fig. 5). With this solution, a tetrahedron intersected by the sweeping surface is always replaced with 17 new tetrahedra. The constraint imposed by S is matched by duplicating the additional points on the intersected edges and by varying their locations along the edges to fit the intersection points. Although this technique provides a straightforward way to implement the replacement, it suffers from two serious drawbacks:

- Since every cut tetrahedron creates 17 new tetrahedra, the mesh density greatly increases in the region around the cut; this mesh refinement effect becomes even more critical when multiple cuts are performed on nearby sections of the represented tissue;
- The mesh returned is no longer a simplicial complex.

Our original solution relies on an ad-hoc tetrahedralization for each cut configuration, stored in a look-up table addressed by the intersected edges as shown in Fig. 6. Each LUT entry stores a set of quadruples, each defining a single tetrahedron to be produced in the output. Each quadruple identifies the vertices of the split tetrahedron (chosen from the vertices of the

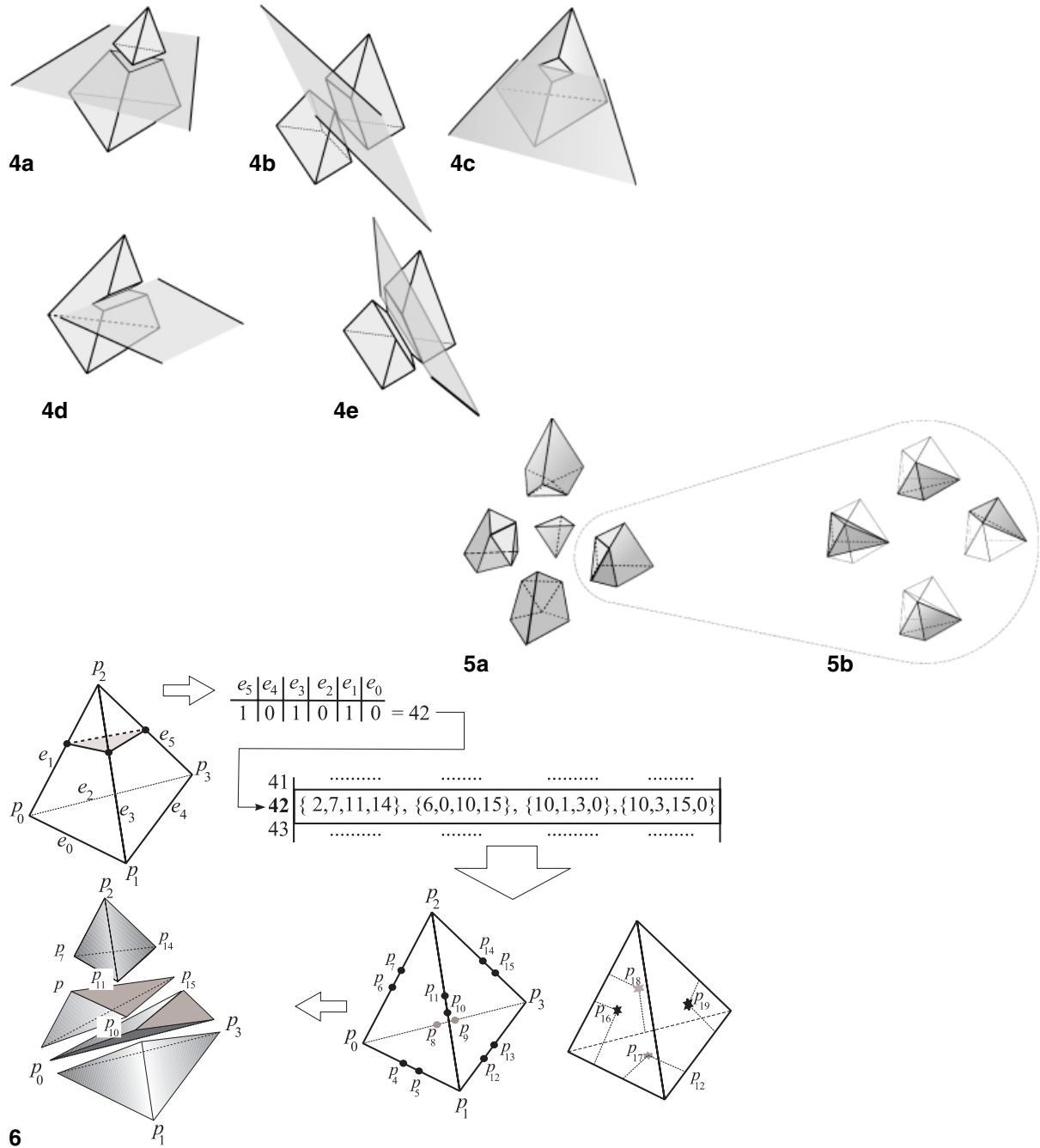
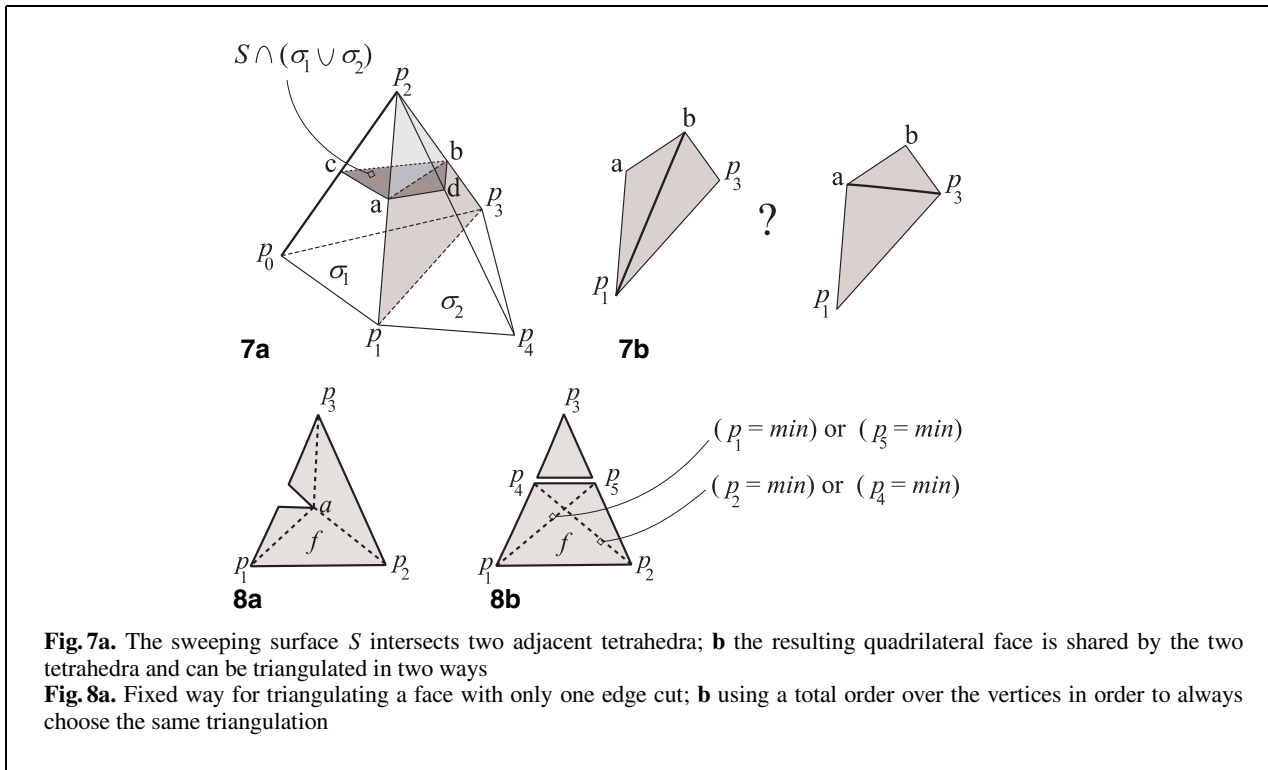


Fig. 4. The five possible cut configurations
Fig. 5a,b. The 1 : 17 splitting scheme: **a** a tetrahedra can be decomposed in to a tetrahedron plus four 7-faces polyhedra; **b** each of these is decomposable into four further tetrahedra
Fig. 6. An example of a cut of type A is shown (*top-left*); the three edges intersected by the sweeping surface produce the code 42 used to access the LUT entry which encodes the new set of 4 tetrahedra (*top-right*); the scheme for interpreting the vertex numbers of the table is shown (*bottom-right*) and the final decomposition into the 4 tetrahedra encoded in the LUT entry 42 is shown (*bottom-left*)



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 Fig. 6. We encode two topologically different intersection points 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, 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 Fig. 4).

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 Fig. 4).

We computed manually the tetrahedralization stored in the LUT for each cut configuration, adding new vertices only on the intersection points between S and the edges of the tetrahedron, and between δS and the faces of the tetrahedron. Simplicial complex properties are, by construction, always preserved.

The condition for a valid replacement can now be written as:

- $R(\sigma)$ is a simplicial complex;
- $R(\sigma) \cap (\Gamma \setminus \{\sigma\})$ is a simplicial complex.

Here Γ is any representation involving σ . Less formally, the problem is on the border of $R(\sigma)$ which is shared with the rest of the mesh, hence the two-dimensional triangulation of such a border is constrained to be the same. Figure 7a shows a practical case: suppose that the sweeping surface intersects two adjacent tetrahedra σ_1 and σ_2 at the points a, b, c, d . After two valid replacements, the two tetrahedra will be replaced by those in the sets $R(\sigma_1)$ and $R(\sigma_2)$ respectively. Since the face (p_1, p_2, p_3) is shared by σ_1 and σ_2 , the new faces (a, b, p_2) and (p_1, a, b, p_3) will be shared by $R(\sigma_1)$ and $R(\sigma_2)$ and hence they have to be triangulated in a unique manner. In the example, two triangulations are possible for the face (p_1, a, b, p_3) , introducing the edge (p_1, b) or the edge (a, p_3) . In general, the problem is to guarantee that a face f shared by the two tetrahedra to be replaced is triangulated in a compatible manner. Observe that there are only two cases:

- Only one edge of f is cut (Fig. 8a);
- Two edges of f are cut (Fig. 8b).

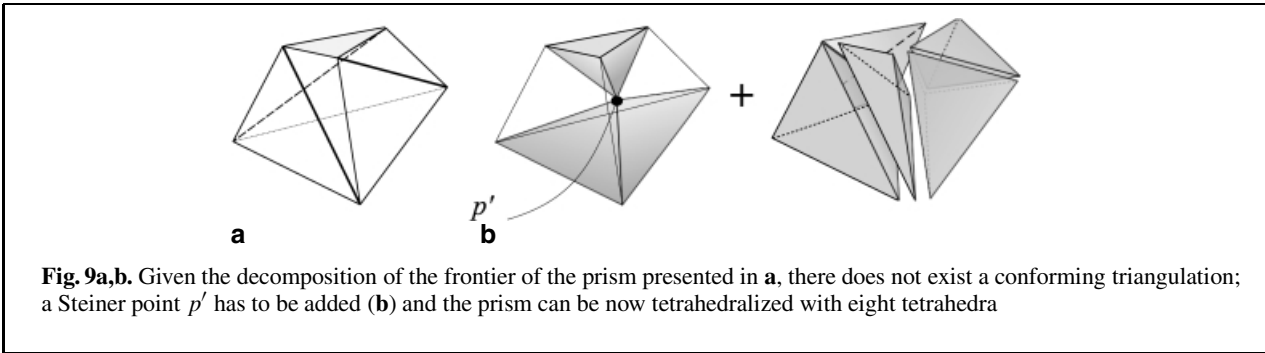


Fig. 9a,b. Given the decomposition of the frontier of the prism presented in **a**, there does not exist a conforming triangulation; a Steiner point p' has to be added (**b**) and the prism can be now tetrahedralized with eight tetrahedra

The first case generates a six-vertex polygon, in which we can use all the diagonals connecting the new vertex a with the three vertices $p_1, p_2,$ and p_3 . The second case (as in the previous example) generates a triangular face and a quadrilateral face. In this case we use a total order on the vertices to choose the same diagonal; this could be the time of insertion into the mesh data structure or the memory address of the object implementing the vertex. In particular we always choose the diagonal incident in the vertex at the bottom of the total order among the four vertices of the face. The result of this analysis is that for the same cut configuration there are multiple tetrahedralizations, according to the different order among the vertices of the newly generated quadrilateral faces. This means that we have to provide two alternative tetrahedralizations for each quadrilateral face. In particular:

- Three quadrilateral faces are created for the configuration A (Fig. 4), hence 2^3 tetrahedralizations are provided;
- Four in case B, hence 2^4 tetrahedralizations;
- No quadrilateral faces in case C;
- One in case D;
- Two in case E, hence 2^2 tetrahedralizations.

In conclusion, our look-up table is 33 entries long (see also Table 1). So far we have defined the constraints to which $R(\sigma)$ is subjected, without having actually defined the set $R(\sigma)$ itself. An open problem is: Is it always possible to define the set $R(\sigma)$ such that the constraints imposed can be satisfied? We proved that this is possible by simply defining each of the 33 tetrahedralizations, with the only exception being that for case A shown in Fig. 9. This is the well-known *un-tetrahedralizable Shonärtdt polyhedron*: it is the only case in which we add a Steiner point in the middle of the prism to create a tetrahedralization matching all three imposed diagonals.

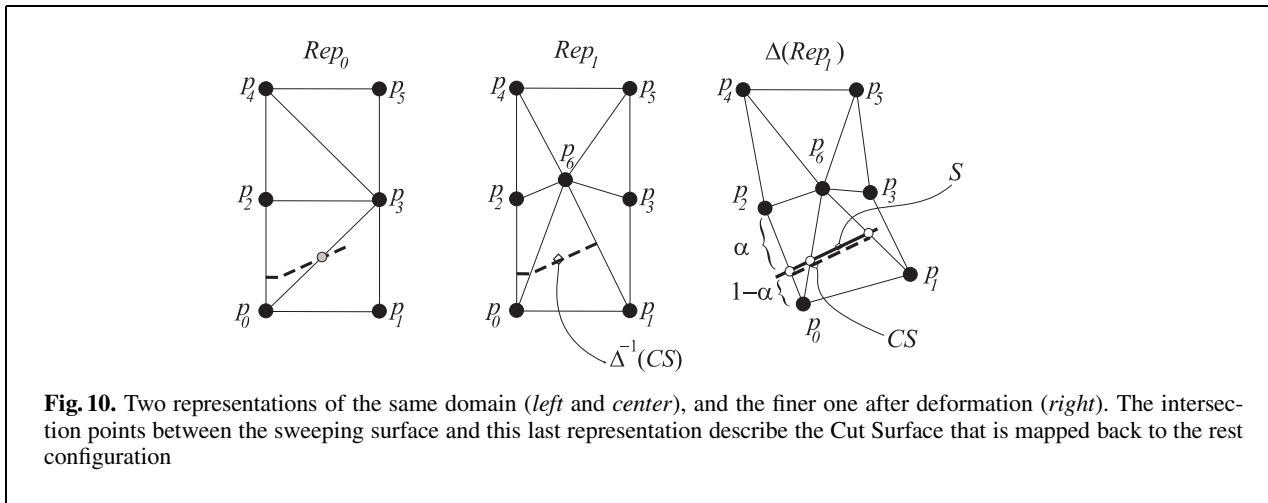
Table 1. Look-up table entries for the different cutting cases. The second row is related to the special case of configuration A which needs a Steiner point

cut config.	no. alternative decompos.	no. tetrahedr. cells	no. rotations/ /mirrorings
A	7	4	4
	1	8	4
B	16	6	3
C	1	6	6
D	2	8	4
E	4	9	8

Table 1 reports the number of tetrahedralizations and the size of $R(\sigma)$, i.e. the number of tetrahedral cells after cell split, for each one of the five cases. Clearly, for each case we have to consider possible rotations and mirroring of the tetrahedron, so, for example, case A covers four cuts, obtained by rotation. The number of possible rotations/mirrorings for each configuration is reported in the last column of Table 1.

5 Extending the algorithm to deformable objects: a time-critical implementation

The algorithm for updating the MT is designed to be naturally implemented in a time-critical style, and in particular the fragments can be independently processed. This property enables us, for example, to update only the fragments used in the current mesh at variable resolution, out of the many encoded in the MT; the remainder of the update task, i.e. updating the complete MT, can be postponed to a later phase (executed only every k time steps). In a context in which the shape of the object varies over time,



there is one more problem for the time-critical implementation, which we introduce with an example. Referring at Fig. 10, let Rep_0 and Rep_1 be two alternative representations of the same object; note that the edges (p_0, p_3) , (p_2, p_3) , and (p_4, p_3) belong to Rep_0 but not to Rep_1 . Now we use the hypothesis that the shape changes over time due to the deformation of the object; therefore, let $\Delta(Rep_1)$ be the representation of Rep_1 after a deformation. At this point, we perform a cut on $\Delta(Rep_1)$; we want to update just the fragments related to the cells of the current representation, and to perform the rest of the updating process later. Since the shape can change over time, the problem is that the spatial location of the surface C at the time of its definition becomes meaningless; in other words, when we need to check for intersection of the edges (p_0, p_3) , (p_2, p_3) , and (p_4, p_3) against the surface C , it will be out of date. If only a rigid-body motion affected the object, this problem would be trivial: we would simply need to express the position of C in object coordinates, that when the object moves or rotates, C moves or rotates as well.

It must be observed that if we take all the triangular faces (segments in the 2D example in Fig. 10) created by cutting the tetrahedra of Rep_1 , we get a triangulation, called CS , of the intersection between C and the object (actually every triangle is duplicated for each side of C ; we keep one of each pair). Furthermore, the position of the vertices of CS can be expressed as a linear combination of the position of the extremes of the edges intersected by S , so we can express CS as a function of the

vertices in $\Delta(Rep_1)$. It follows that using the original position of the vertices, we can map CS to the rest configuration (Ganovelli 2000). Referring again to the figure, we see that CS is defined by the three points P_{02} , P_{06} , and P_{16} ; each of these points is associated with the coefficient of the linear combination (α is shown in the case relative to P_{02}) giving its position. Therefore the leftmost point of the surface $\Delta^{-1}(CS)$ is linearly interpolated from the positions of p_0 and p_2 at the rest configuration.

6 Results

Figure 11 shows a run of the algorithm on a mechanical piece. The first step is to place the tool used to interactively define the cut: we use a disk in the example (Fig. 11b). Figure 11c shows the two pieces arising from the split of the original piece with the new tetrahedra (rendered using a darker gray). Two independent MTs have been produced to represent the resulting two pieces. In the last image (Fig. 11d) the same pieces are shown from a different viewpoint and are represented with two different levels of detail.

Table 2 reports results from cut operations. In particular, the third row of the table contains the data for the snapshots of Fig. 11. The columns of the table report 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 LUT-based solution, then the times

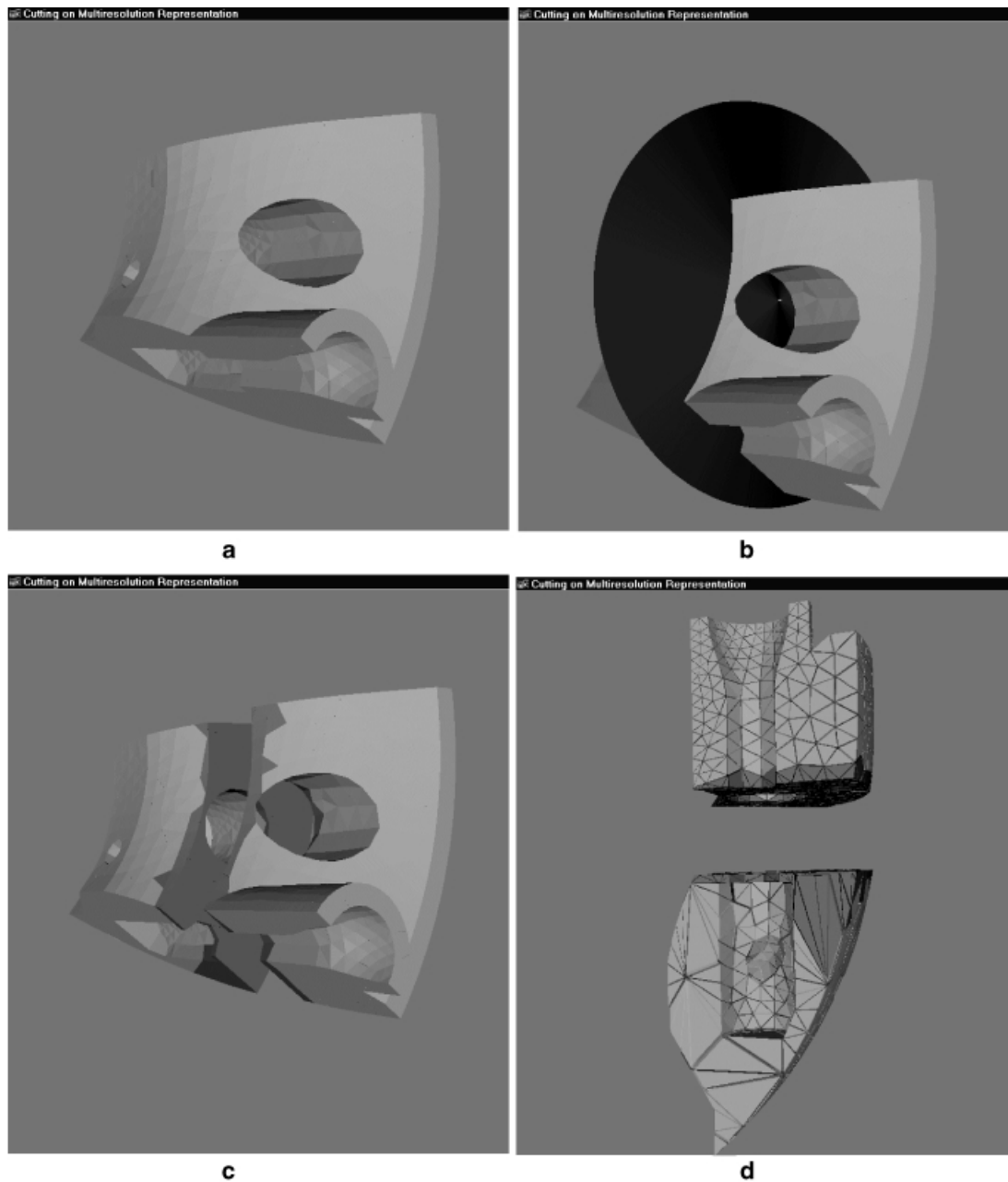


Fig. 11. Snapshots of the application window. The MT associated with the piece is split into two independent MTs

for splitting the tetrahedra, the time for updating the DAG, and the total time required (all times are expressed in milliseconds and are evaluated on a PC Pentium II, 400 MHz, 256 MB RAM).

When using our solution in a virtual surgery application, two points relating to computational efficiency have to be considered:

- The typical way to simulate the cut operated with a virtual scalpel is to detect its position in consecutive time steps and to consider the surface swept (the scalpel is assumed to be a segment) as the cutting surface. The times resulting from our tests are related to a cut that involves many tetrahedra, such that in some cases the object is split into two

Table 2. Examples of cuts. The original MT is composed of 2896 vertices, 37 062 tetrahedra and 1542 nodes (fragments). Each row corresponds to a different cut. In the first two columns, the number of fragments and tetrahedra involved in the cut is indicated; the third column indicates the number of cells created by splitting; the last three columns report: the time for splitting and replacing tetrahedra, the time for updating the DAG; and the total time

Cut complexity		New cells introduced		split cells	Times (ms.)	
no. fragments	no. input cells	no. cells 1:17	no. cells our LUT		update DAG	total
36	523	8891	2394	63	156	219
148	1813	30 821	8300	250	109	359
205	2520	42 840	11 612	277	117	394
245	3122	53 074	14 316	344	141	485
528	6552	111 384	29 918	531	344	875

parts. It would correspond to the not very realistic case of trespassing the entire object with the scalpel in two consecutive time steps. Note that the running times in Table 2 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 all the tetrahedra in the DAG (i.e. even the tetrahedra which are not part of the current variable-resolution mesh used in the simulation and visualization) and the DAG structure. However, since other operations also have to be performed 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 should be realized. Thanks to the extension presented in Sect. 5 it can be obtained by processing those fragments which are currently visualized first, and postponing the others to later time steps, with the only restriction being that the resolution cannot change until all the fragments have been processed and updated.

be further optimized, experiments have shown the efficiency of the solution proposed in terms of both reduced fragmentation of the tetrahedral representation, and time. Our next step will be to adopt this approach in a prototypal system which implements physical simulation.

Simulating a cut is a fundamental feature of a virtual surgery system, but the inverse operation, the suture, should also be treated. More research is needed on this issue.

We conclude by introducing a straightforward application of this work, which is not connected with virtual surgery. The MT scheme allows us 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 that region, thus 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. For a very large dataset this can be a problem, because of RAM limitations. With our algorithm, we can easily create an MT which corresponds to the specified region of interest, omitting everything outside such that region.

7 Conclusion and future work

This work has been inspired by the need to perform cuts on meshes representing soft tissues in a multiresolution framework. Both multiresolution and cuts have been previously introduced for deformable object modeling: unfortunately their use was mutually exclusive. We have defined an approach to fill this gap in the case of tetrahedral representations, based on the adoption of the MT multiresolution scheme. Although the implementation could

References

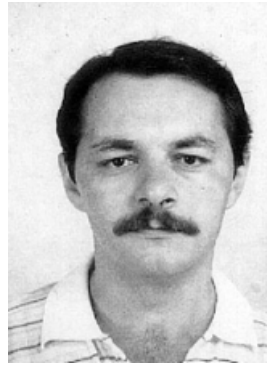
1. Berkley J, Ganter M, Weghrost S, Gladstone H, Raugi G, Berg D (1999) Real-time finite element modelling with haptic support. In: Proceedings of ASME Design Engineering Technical Conference, pp 287–296
2. Bielser D, Maiwald V, Gross M (1999) Interactive cuts through 3-dimensional soft tissue. (Eurographics '99) Comput Graph Forum 18:C31–C38
3. Cugini U, Bordegoni M, Rizzi C, Angelis FD, Prati M (1999) Modelling and haptic interaction with non-rigid materials. (Eurographics'99) Comput Graph Forum 18:1–20

4. De Floriani L, Magillo P, Puppo E (1998) Efficient implementation of multi-triangulations. In: *Proceedings IEEE Visualization '98*, ACM Press, pp 43–50
5. De Floriani L, Puppo E, Magillo P (1997) A formal approach to multiresolution modeling. In: Klein R, Straßer W, Rau R (eds) *Geometric modeling: theory and practice*. Springer, Berlin Heidelberg New York, pp 302–323
6. Debunne G, Desbrun M, Barr A, Cani M (1999) Interactive multiresolution animation of deformable models. In: Magnenat-Thalmann N, Thalmann D (eds) *Eurographics Workshop on Computer Animation and Simulation '99*. Springer, Berlin Heidelberg New York, pp 133–144
7. Debunne G, Desbrun M, Cani M-P, Barr A (2000) Adaptive simulation of soft bodies in real-time. In: *Proceedings of the Conference in Computer Animation CA '00*, pp 15–20
8. Delingette H, Cotin S, Ayache N (1999) A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In: *CAS '99 Proceedings*, pp 70–81
9. Ganovelli F (2000) *Multiresolution and cuts in deformable object modeling*. PhD thesis, Dept. of Computer Science, Università degli Studi di Pisa
10. Ganovelli F, Cignoni P, Scopigno R (1999) Introducing multiresolution representation in deformable modeling. In: Zara J (ed) *SCCG '99 Conference Proceedings*, Budmerice (Slovakia), pp 149–158
11. Gibson S, Mirtich B (1997) A survey of deformable modeling in computer graphics. Technical Report TR-97-19, Mitsubishi Electric Research Laboratories
12. Hutchinson D, Preston M, Hewitt T (1996) Adaptive refinement for mass spring simulations. In: Boulic R, Hédrón G (eds) *7th Eurographics Workshop on Animation and Simulation*
13. Magillo P (1999) *Spatial operations on multiresolution cell complexes*. PhD thesis, Università degli Studi di Genova
14. Nienhuys H-W, van der Stappen F (2000) Combining finite element deformation with cutting for surgery simulations. In: *Proceedings of Eurographics 2000, short presentations*, pp 43–51
15. Puppo E (1996) Variable resolution terrain surfaces. In: *Proceedings Eight Canadian Conference on Computational Geometry*, Ottawa, Canada, pp 202–210

Photographs of the authors and their biographies are given on the next page.



FABIO GANOVELLI is a PhD student at the National Research Council of Pisa. His research interests include deformable object modelling, collision detection, multiresolution and isosurfaces extraction. He received an advanced degree in Computer Science (Laurea) in 1995 from the University of Pisa.



CLAUDIO MONTANI is a research director with the Istituto Scienza e Tecnologia dell'Informazione (formerly I.E.I. – CNR) of the National Research Council in Pisa, Italy. His research interests include data structures and algorithms for volume visualization and rendering of regular or scattered datasets. Montani received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1977. He is member of IEEE.



PAOLO CIGNONI is research scientist at the Istituto Scienza e Tecnologia dell'Informazione (formerly I.E.I. – CNR) of the National Research Council in Pisa, Italy. His research interests include computational geometry and its interaction with computer graphics, scientific visualization, volume rendering, simplification and multiresolution. Cignoni received in 1992 an advanced degree (Laurea) and in 1998 a PhD in Computer Science from the University of Pisa.



ROBERTO SCOPIGNO is senior research scientist at the Istituto Scienza e Tecnologia dell'Informazione (formerly CNUCE – CNR) of the National Research Council in Pisa, Italy. He is currently engaged in research projects concerned with scientific visualization, volume rendering, web-based graphics, multiresolution data modeling and rendering, 3D scanning and applications of 3D computer graphics to Cultural Heritage. Scopigno received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1984. He is member of IEEE, Eurographics and Siggraph.