The Visual Computer manuscript No.
(will be inserted by the editor)

Nico Pietroni · Fabio Ganovelli · Paolo Cignoni · Roberto Scopigno

Splitting Cubes: a Fast and Robust Technique for Virtual Cutting

Abstract This paper presents the Splitting Cubes, a fast and robust technique to perform interactive virtual cutting on deformable objects.

The technique relies on two ideas. The first one is to embed the deformable object in a regular grid, to apply the deformation function to the grid nodes and to interpolate the deformation inside each cell from its 8 nodes. The second idea is to produce a tessellation for the boundary of the object on the base of the intersections of such boundary with the edges of the grid. Please note that the boundary can be expressed in any way, for example it can be a triangle mesh, an implicit or a parametric surface. The only requirement is that the intersection between the boundary and the grid edges can be computed. This paper shows how the interpolation of the deformation inside the cells can be used to produce discontinuities in the deformation function, and the intersections of the cut surface can be used to visually show the cuts on the oject.

The Splitting Cubes is essentially a tessellation algorithm for growing, deformable surface and it can be applied to any method for animating deformable objects. In this paper the case of the mesh-free methods (MMs) is considered: in this context, we decribed a practical GPU friendly method, that we named the Extended Visibility criterion, to introduce discontinuities of the deformation.

Keywords Physically based modelling · Three-Dimensional Graphics and Realism · Animation

1 Introduction

Interactive virtual cutting and tearing of deformable objects are mandatory for surgery simulation. In the last decade, several solutions have been proposed. Most of these methods adopt a mesh-based representation (either of the volume or of the surface) that can be dynamically adapted to animate topological changes. The main problems of these mesh-based approaches concern the fragmentation of the representation in proximity of the regions being cut and/or the accuracy of the representation of the cut. If the mesh describing the geometry is also used as a partition of the object in finite elements for numerical simulation, then the quality of re-meshing affects the stability of the simulation.

More recently, the so-called *mesh-free* methods (**MMs**), traditionally used in fluids simulation, have been introduced in the computer graphics community to model solids [15]. MMs approximates the physical quantities (strain, stress velocity etc.) from those sampled at specific locations. MMs avoid the problems related to re-meshing after a cut and naturally provide the continuity of the physical

Nico Pietroni
Visual Computing Lab, ISTI CNR Pisa and Endocas Center For Computer Assisted Surgery Pisa
email: nico.pietroni@isti.cnr.it

Fabio Ganovelli, Paolo Cignoni, Roberto Scopigno
Visual Computing Lab, ISTI CNR Pisa,Italy
email: name.surname@isti.cnr.it

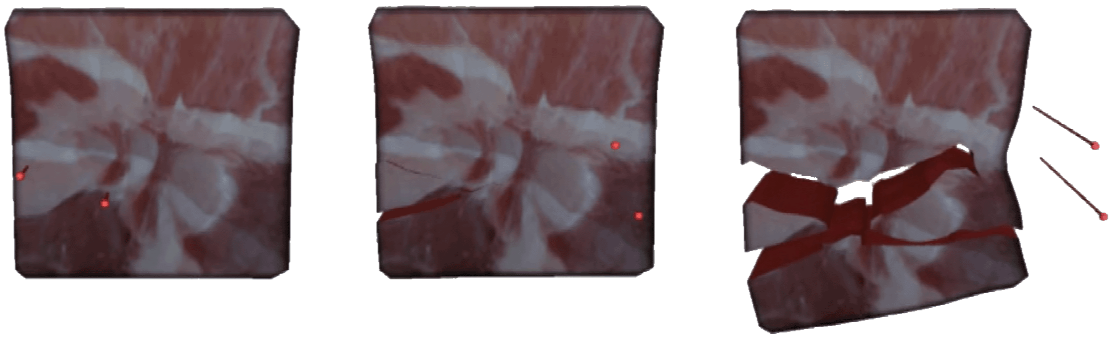


Fig. 1 Frames captured during an interactive simulation. Two blades (clearly visible in the right most frame) are rotating counterclockwise and translating from left to right.

quantities involved in the simulation (such as *strain*). On the other hand MMs also pose problems such as implementing the Essential Boundary Conditions, representing the surface and modifying the model to represent discontinuities [23]. The problem of modelling cut and fractures in offline simulations has been extensively investigated in the field of applied mechanics. On the other hand, for the interactive setting there is still research to do towards a stable and efficient solution, although some pioneering work exists. There are two main problems to solve in order to model cuts in MMs: how to provide a representation of the object's surface that can be updated on-the-fly and how to update the physical simulation to reflect the discontinuities introduced by the cuts. In this paper we introduce a novel solution to both these problems:

- We introduce the *Splitting Cubes*, a new algorithm which provides a dynamic tessellation of an evolving surface embedded in a deforming space. The key idea is to embed the object in a regular grid and to encode a tessellation of its surface in terms of intersection of the surface with the edges of the grid. The position of the tessellation vertices is interpolated by the grid nodes so that it is possible to implement discontinuities at a sub-cell level.

- We introduce the *Extended Visibility criterion*, a new way to handle discontinuities with MMs. With respect to the existing solutions, the Extended Visibility criterion guarantees a smoother adaptation of the system and it can be efficiently implemented harnessing the GPU power.

It is important to note that, although these contributions can be adopted to implement cuts with MMs, they are mutually independent and could be individually be applied to other cases. The Splitting Cubes is essentially a tessellation algorithm for deformable surface that only relies on a generic deformation function, not necessarily obtained with MMs, and on a description of the object's surface (geometric, parametric or implicit). Similarly, the Extended Visibility Criterion is a contribution to the field of MMs and only uses the proximity relation between samples and a tessellation of the crack surface, no matter how it has been created.

Note that we do not propose a new method for physically based modelling of deformable objects. Instead, we propose a new method to represent cuts on deformable objects which is independent of the physical model used.

The paper proceeds as follows: in Section 2 we give a general definition of the problem of cutting and briefly review the approaches proposed so far. In Section 3 we introduce the basic concepts of the Splitting Cubes technique and in Section 4 we describe our Extended Visibility Criterion approach. Results are reported in Section 5 and conclusions and directions for future work are commented in Section 6.

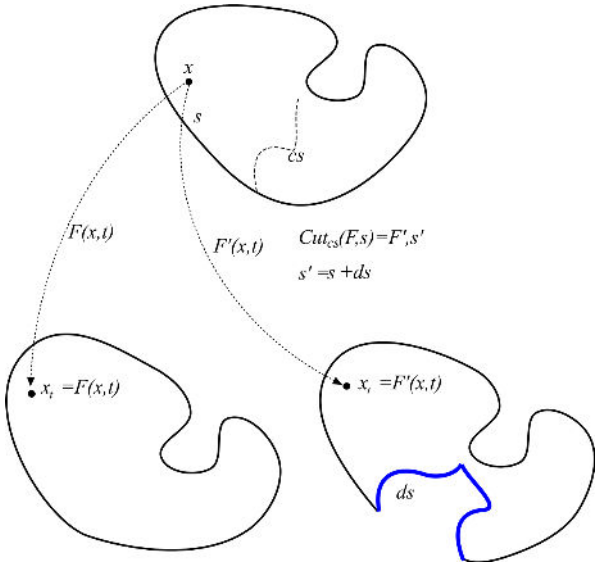


Fig. 2 Definition of cut as a function that takes a deformation function F and a cut surface cs and returns a new surface s' and a new deformation function F'.

2 Background

Many solutions have been proposed to the problem of virtual cutting, using different methods for physical simulation and rendering.

For the sake of generality, we introduce the problem in abstract terms. We characterize a deformable object as a time dependent function $F : \Omega \times T_{ime} \rightarrow I\!R^3$ and a *description* of its surface s, as shown in Figure 2. The function F gives, for each point of the object domain at rest shape Ω, its position at a given time t. F is usually at least C_0 continuous except on the boundary of Ω. A *cut* is a function that modifies F and s on the base of a *cut surface* inside the volume Ω: $Cut_{cs}(F, s) = (F', s')$. In this terms, the problem of virtual cutting can be expressed as the problem of defining the function Cut.

Most of the methods for real time interaction with deformable objects deal with *mesh-based* models. In these models, the refresh rate of the physical system is linearly related to the number of primitives of the mesh. Moreover, the stability of dynamic solvers is strongly influenced by the quality of the elements in the mesh. Therefore *mesh-based* methods are often focused on how to produce an accurate representation of the cuts while minimizing the number of primitives created and taking into account their quality. In the next two sections we will concisely review the literature on interactive cuts applied on mesh based and on mesh-free models.

2.1 Cuts on mesh based models

Delingette et. al. [6] proposed a hybrid model where the portion of the object that is supposed to be cut is modeled with the *tensor mass*, a scheme similar to the mass spring system where the forces are derived per tetrahedron from the displacement of its four vertices, while the rest of the mesh is modeled with a more accurate FEM. In their approach the cut is implemented by removing the tetrahedra touched by the cutting tool. This method avoids the creation of new primitives, but introduces the serious drawbacks of poor visual feedback as well as the loss of volume (see Figure 3.(b)).
In the solution proposed by Nienhuys et al. [16] the mesh vertices closer to the cut are snapped onto the cut surface and duplicated to open the cut (see Figure 3.(c)). This method does not create new tetrahedra and can be coupled with a FEM simulation, since the updating of the stiffness matrix can be done on-the-fly.
Several authors used re-meshing to adapt the tessellation to represent the cut surface [26,5,12] by using a triangulated surface and splitting the triangles intersected by the cutting tool. In case of volume

representation with tetrahedral meshes, Bielser et al. [4,3] and Ganovelli et al.[7] used dynamic re-meshing of the intersected tetrahedra to adapt the new boundary of the model to the cut surface (see Figure 3.(d)). Re-meshing provides an accurate representation of the cut surface, although it produces mesh fragmentation that can be only partially alleviated by enhancing the re-meshing strategy with on-the-fly edge collapse operations [8] or by adopting a combination of these techniques [24].
O'Brien et al. proposed a solution for modeling brittle and ductile fractures [18,17] in off-line simulations. They used continuum mechanics equations to derive the crack surface and mass lumping to provide an explicit integration scheme. In their method re-meshing is used to accurately represent the crack surface, since "approximating it with the existing element boundaries would create undesirable artifacts" [18].
Recent solutions decouple the simulation from the representation. In [14] each tetrahedron of the mesh can be decomposed on the base of which of its edges are crossed by the cut surface, but such decomposition does not replace the orginal tetrahedron for the physical simulation. If the cutting generates disconnected components, the tetrahedron is duplicated. While in [14] the tetrahedron can be decomposed at most in 4 components (one for each node), in [22] this idea extended by allowing the tetrahedra to be split any number of time, always considering the intersection of the crack surface with the current decomposition, and not only with the 6 edges of the original tetrahedron. In this manner the objects can be cut in pieces arbitrarily small, at the price of generating polyhedra with any number of faces (which all need to be tested for intersection and collision detection).

2.2 Cuts with Mesh free Methods

MMs are methods to solve partial differential equations numerically without the support of a partition of the domain in finite elements. Here we give some background info necessary to follow the rest of the paper, the reader may refer to [23] for a complete monograph. In MMs the value of an unknown variable at a generic point x in the domain is approximated by the value of a number of samples x_i, termed *phyxels* in [15]. If the unknown variable is the deformation, we can write:

$$F(x,t) = \sum_{i \in P} \Phi_i(x) F(x_i, t) \tag{1}$$

where $\phi_i(x)$ are continuous functions, called *shape functions*. The shape functions are weighted by a function of the distance between the phyxel and the point to approximate, written as $w(x, x_i, r_i)$ (incorporated in $\Phi_i(x)$), which rapidly decays with the increase of $\|x - x_i\|$. The radius of influence r_i of the shape function is typically chosen to include a constant number of neighbors (e.g. 10 in [15]) that will be the phyxels directly influenced by (and influencing) phyxel i.
MMs shape functions provide a higher order of continuity with respect to FEM shape functions but also pose some difficulties. Given that the MMs shape functions do not verify the *Kronecker delta* property like the FEM shape functions do [1], it is more difficult to impose the essential boundary conditions, e.g. to move a phyxel to a desired position. Similarly, special care needs to be taken to model discontinuity inside the material, which is done either by enriching the shape function or, more commonly, changing the weight function to loosen the mutual influence between phyxels on different sides with respect to the discontinuity. Finally, the meshfree method do not naturally provide a representation of the boundary.

Adapting the surface in mesh free methods.
In the solution proposed by Pauly et al. [20] the surface of the deformable object is dynamically sampled with *surfels* represented as oriented elliptical splats. In order to show sharp features, which are always created by cutting, the surfels overlapping a crease can be clipped against a plane lying on the other side of the crease [21].

In their model a crack is codified by a sequence of phyxels (called *crack nodes*) which represents the propagation front of the crack. For cracks starting from the surface (e.g. when a cut is being made), the first and last node of the sequence lie on the surface while for cracks generating inside the volume the front is circular. Every time a new crack node is added to the sequence, i.e. every time the

[1] The Kronecker delta property refers to the fact that the shape function $\Phi_i(x_j) = 1.0$ if $j = i$ and 0 otherwise

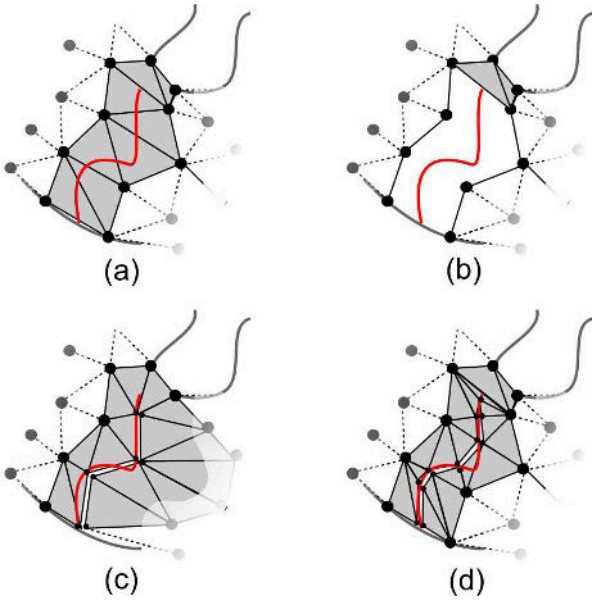


Fig. 3 Techniques to implement cuts in mesh based models. (a) Portion of a triangulation with a cut surface (in red) (b) Removing elements (c) Snapping vertices on the cut surface (d) Remeshing

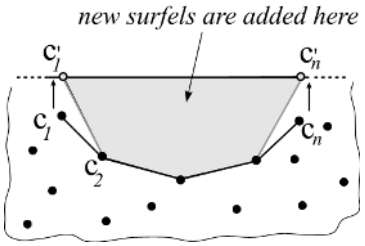


Fig. 4 Example of crack front.

front propagates, new surfels are added to represent the two new pieces of surface (see Figure 4). This technique avoids the classical problems of the mesh-based methods, i.e. fragmentation and degeneracies due to re-meshing. On the other hand the crack fronts can split and merge and these events need to be handled explicitly to maintain the topology of the crack front(s) consistent.

An alternative approach has been proposed by Steinemann et al. [25], where the surface is represented by a triangle mesh. When a cutting tool penetrates the object, the cut surface is triangulated and used to update the current object's surface with the new pieces of sheets. These new sheets are created by triangulating the portion of the splitting surface inside the volume, triangle by triangle. As for the previous solution, the branching and merging of crack fronts has to be handled explicitly. Compared to the point sampled method described above, the use of a triangle mesh give advantages in terms of rendering speed, but if multiple cuts are executed in the same region triangle fragmentation and degeneracies may occur, causing degradation in performance and in the quality of the intersection tests.

Adapting the physical simulation

As previously stated, a cut introduces a discontinuity in the deformation function. In MMs this discontinuity can be computed by altering the weights of the the shape functions, which can be done in various ways.

One straightforward way is the *visibility criterion* proposed by Belytschko et al. in [2] and consists of zeroing the value of the shape function $\Phi_i(x)$ in those points from which the phyxel i is not visible, i.e. if the segment $\overline{x\ x_i}$ crosses the newly created surface. Although very simple to implement, this method introduces also an undesired discontinuity at the horizon line (see Figure 12.a) that affects

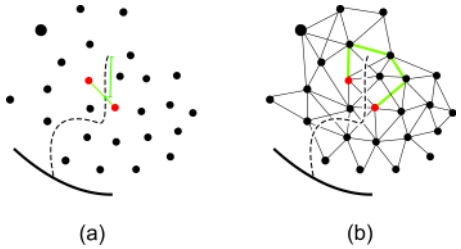


Fig. 5 (a) Lengthening the distance by transparency method. (b) Approximation of refraction method by exploring the graph of phyxels.

convergence and stability.
In the *diffraction method* [1,19] the distance between two points is defined as the shortest curve not intersecting the discontinuity line (see Figure 12.b).
An approximated but interactive version of this approach is also used by Steinemann et al. in [25]. In a preprocessing phase, they build a connectivity graph on the phyxels by adding an edge for each couple of phyxels (i, j) such that $w(d(i, j)) \neq 0$. Then the distance between two phyxels i and j is always taken as the shortest path in the connectivity graph. When a cut surface is defined, the arcs intersecting the cut surface are removed and the shortest paths between the phyxels in the neighborhood are recomputed (see Figure 5). The *transparency method*, proposed in [1], considers the intersection between $\overline{x\ x_i}$ and the cut surface and the distance of the intersection point to the closest border of the cut surface (see Figure 5.a). This method is also used in [20] in a non-interactive simulation.

3 The Splitting Cubes Algorithm

The Splitting Cubes algorithm is a technique for providing a tessellation of an evolving surface embedded in a deforming space. In the specific case treated in this paper the surface evolves when a cut exposes new parts of the object boundary, and when the space deforms.

The key idea of the Splitting Cubes algorithm is to embed the object in a regular $3D$ grid whose nodes are moved according to the deformation function F, and to interpolate F inside each cell from its 8 nodes. This scheme allows us to implement discontinuities of the deformation inside a cell, by varying the interpolation values of the nodes, depending on which edges are cut. We introduce the details with a practical $2D$ example. Figure 6.a shows a tessellated surface crossing a few cells of the regular grid (in $2D$). The cyan arrows leaving from a vertex of the tessellation show the dependencies of that vertex on the cell nodes, i.e. from which nodes we compute its position. The red curve shows the intersection of a cutting tool path with one edge of the embedding grid. The tessellated surface is defined cell by cell on the base of the configuration of cut edges (position and normal of the intersections), similarly to what is done in the various extensions of the Marching Cubes approach [13,28] that exploit hermitian data [11, 10]. However, the cut shown in Figure 6.b would generate an invalid configuration for the standard Look Up Table (LUT) of the Marching Cubes both for cell $CellA$ and cell $CellB$ (one edge of the cell is intersected). On the contrary, the Splitting Cubes algorithm includes these configurations. The reason relies in the nature of the cut surface. In the literature the cut surface is regarded as the surface swept by the cutting tool, which is identified with a segment. We use a more topological definition to explain our technique: the cut surface is the boundary of a protrusion of the space surrounding the object. In other words, when a cutting tool penetrates into the object, it actually extends the empty space into the object, and the cut surface is the boundary of that portion. Although at rest shape the volume of this protrusion is zero, its boundary (the blue curve in Figure 2) is topologically well defined and can be tessellated, which is exactly what the Splitting Cube does by sampling the cut surface on the cells edges and using these points to define a tessellation.

The tessellation is defined individually cell by cell. Figure 6.b shows the tessellation for the configurations of cells $CellA$ and $CellB$. We can see that the cut generates two vertices on the edge and one inside the cell $CellB$. Furthermore, the dependencies of the vertices inside the two cells have been changed to reflect the cut.

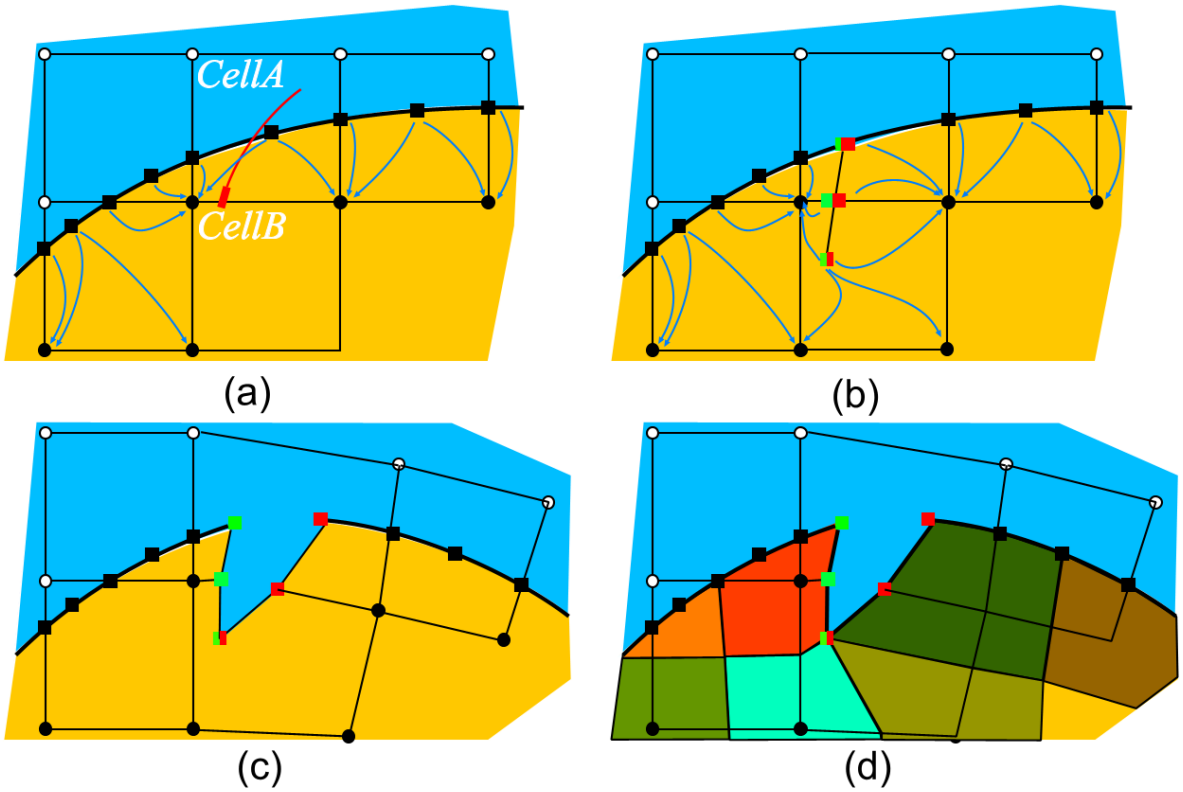


Fig. 6 The Splitting Cubes idea. (a) The object is embedded in a regular grid, the vertices of the tessellated surface depends on the grid nodes. (b) A cut surface crosses an edge and changes the configuration of cells CellA and CellB. (c) The new surface in the deformed space. (d) Dual interpretation: cuboidal portion of material are lumped to the grid nodes.

The Splitting Cubes LUT contains all the 2^{12} possible configurations determined by cuts on cell edges, and for each triangulation it specifies the dependencies of the vertices from the cell nodes. Figure 6.(d) shows a dual interpretation of the Splitting Cubes where every node represents an amount of material and the material of two adjacent nodes is continuous if and only if the corresponding edge is not cut. We give also this interpretation to show how the Splitting Cubes could be seen as the version of the Virtual Node algorithm [14] on regular grids instead of tetrahedra. Also, we will use this interpretation for explaining the construction of the LUT (see Appendix A).

Figure 7 shows the six configurations for a cell in the $2D$ case. For each configuration a tessellation of surface inside the cell is given. The cyan arrows leaving the vertices and pointing to the cell nodes show the dependencies.
We start considering the $2D$ case, where the cells are quadrilaterals (beside, these are also the configurations for the faces of a $3D$ cell). The first column shows the configuration in the parametric domain while the second shows a possible deformation of the cell with the vertex-node dependencies. The number next to the case letter indicates how many equivalent configurations are obtained by rotation or mirroring.
The configurations **B**-**F** are tessellations of the cut surface as derived by the cut edges. Note that each cut edge will always create two vertices, called *edge-vertices* from now on, and that each edge-vertex always depend on only one of the two extremes of the edge. This choice reflects the discontinuity of the deformation function F along the edge and will allow the two vertices to be taken apart in the deformed space.

Similarly, the vertex in the middle of the face, called *face-vertex*, is replicated for each connected component and only depends on the nodes of its connected component. A connected component is defined as a portion of the cell where every pair of points can be connected by a curve without intersecting the cut surface.

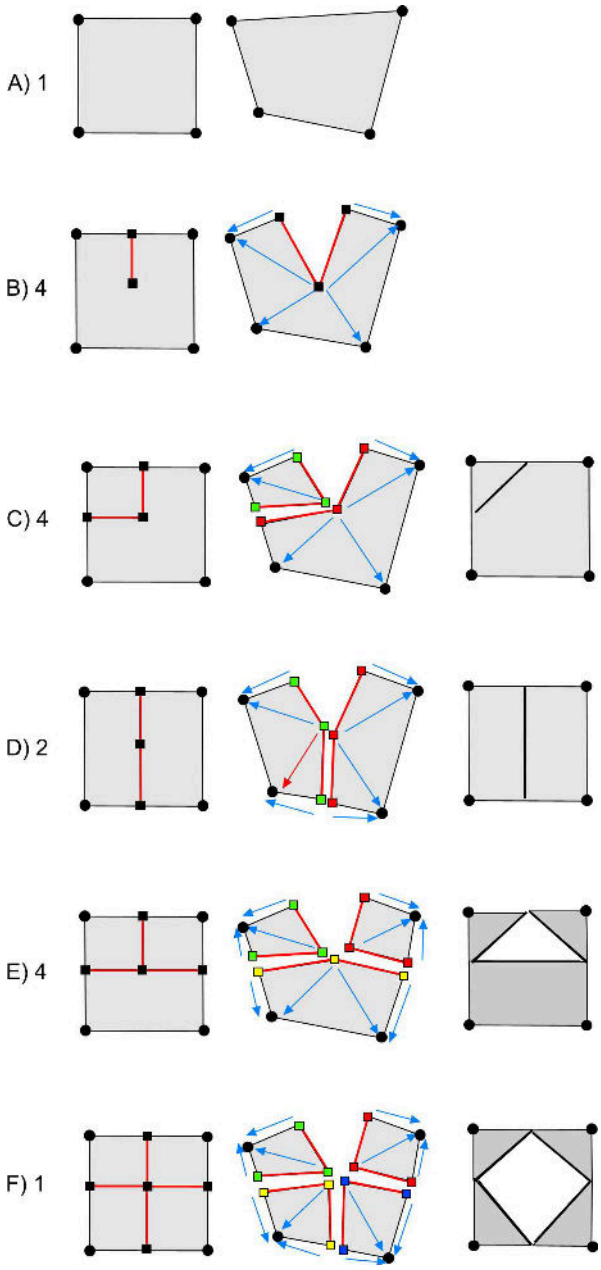


Fig. 7 The six configurations for a face of the splitting cube (i.e. the 2D case). We show: the configurations at rest shape (leftmost column); the configurations at a given deformed shape (center column); the need for the internal face vertex to avoid volume loss (right most column).

Note that, except for the case B, the *face-vertex* would not be necessary to build the triangulation. On the other hand, introducing face-vertices is necessary to preserve the amount of material, otherwise cuts will result in a reduces mass (see cases E and F in the figure).

The configuration for the $3D$ cell are derived by extending the $2D$ case. Let us consider a cell with one edge cut, resulting in two faces with a **B** configuration as shown in Figure 8.(a). We can build a quad with the edge-vertex, the two face vertices and a *center vertex* placed inside the cube. We build such a quad for each one of the cut edges and then find our connected components, duplicate the vertices and assign the dependencies accordingly. Figure 8.(b) and 8.(d) shows a case with 5 edges cut and its triangulation, respectively. In this case two connected components are found.

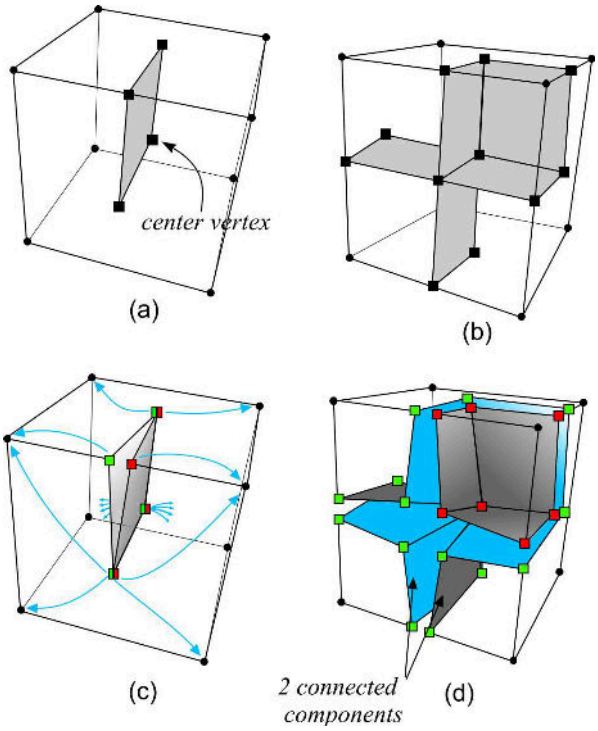


Fig. 8 Two examples of cuts. (a-b): The quads created by cutting 1 and 5 edges. (c-d): the resulting tessellation.

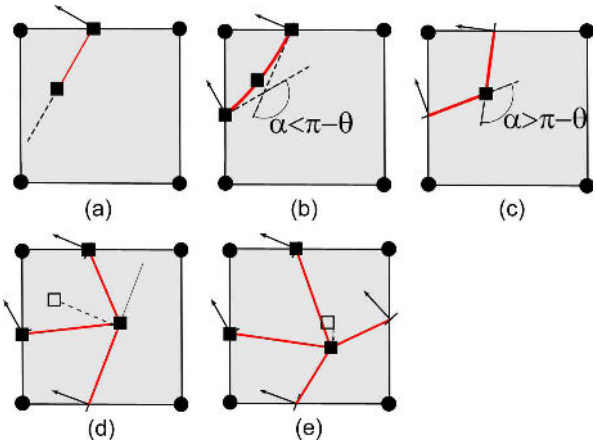


Fig. 9 The figure shows how the vertices of the tessellation are derived from the intersection o the cut surface with the edges of the cell and the normal to the cut surface at the crossing point.

The tessellation and the dependencies are computed once for all and stored in a Look Up Table with 2^{12} entries.

3.1 Position of the vertices

While the connectivity of the vertices added by a cut is stored in the LUT, their position has to be found on-the-fly. For the edge vertex the choice is trivially the cutting point along the edge. The position of face points is less obvious to find. Our goal is to provide a tessellation that mimics the cut surface inside the cell, so we cannot use simplistic solutions as the center of the face or the average of the edge vertices. Instead we also take into account the normal of the cut surface at the edge-vertices, that we

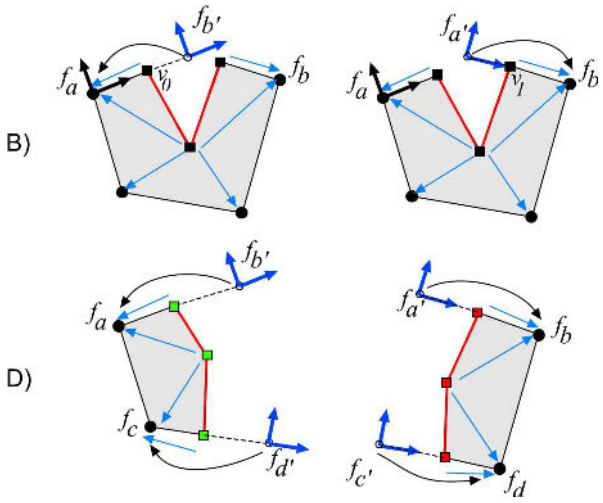


Fig. 10 Two types of cut (case B and case D). The frames rendered in blue are virtual and depend on the frame pointed by the arrow.

obtain from the movement of the tool. This normal and the relative edge-vertex define a plane the we call *Cut Plane*. In other terms, a cut plane is the approximation of the cut surface around a edge vertex. For the case **C** and **D**, where two edges are cut, we use the approach described in [11]: if the angle formed by the two planes passing through the edge vertices is close to π, we assume the two edges have been cut with a smooth movement and place the faced vertex in the middle of the Bezier curve defined by the edge vertices and the intersection of their associated cut planes (see Figure 9.(b)). Otherwise we place the vertex exactly at such intersection point, showing a sharp feature (see Figure 9.(c)). The cases **E** and **F** would clearly require a tessellation with more degrees of freedom for representing the cut surface exactly. We use a strategy which leads to little or no visible artifacts in the assumption that the cut is done by a single tool (which is not restrictive in most scenarios). The idea is that in a single movement of the tool, no more than two edges will be cut, producing a configuration from **A** to **D**. Any further cut will find the face vertex already placed, so we redefine (move) the face vertex by projecting it into the plane defined by the new cut plane, as shown in Figure 9.(d) and Figure 9.(e).

Similar considerations hold for the position of the central vertex. The first time the cell is cut, if the cut does not split the cell in two parts, the central vertex is positioned in the average position among the face vertices, otherwise it is placed so that it minimizes the sum of the squared distances from all the planes by using a quadric metric as in [9]. Again, for any further cut we project the position onto the new cut plane.

3.2 Interpolation inside a cell

As previously stated the space inside the cell is deformed accordingly to the cell nodes.
We attach a reference frame to each node i, $f_i = (A_i, O_i)$ where A_i are the three axes and O_i the origin. Given the deformation function F, the frame f_i at time t is found as $f_{i_t} = ((J_{F_t}^{-1})^T(A_i), F(O_i))$, where J is the Jacobian of F. The position in the deformed space of a generic point p is found as:

$$\sum_{j \in cell} a_j p_j f_{j_t} \tag{2}$$

where p_j is the projection of p on the frame f_i and a_j are the scalar coefficients of the trilinear interpolation. This interpolation scheme uses all of the 8 nodes of a cell, while tessellation vertices depend only on a subset of them. In the example in Figure 10), the position of the edge vertex v_0 is interpolated from f_a and f_b, since the coefficient of the trilinear interpolation are 0 for the other 6 frames. However, since we want to represent the discontinuity of F along the edge, we impose that v_0 only depends on f_a while the frame f_{b_t} is replaced by the frame $f_{b'_t}$ which is computed by the function $F'(b) = F(a) + (b - a)\ J_{F_t}(a)$, i.e. the deformation function in b as approximated by its value in a by

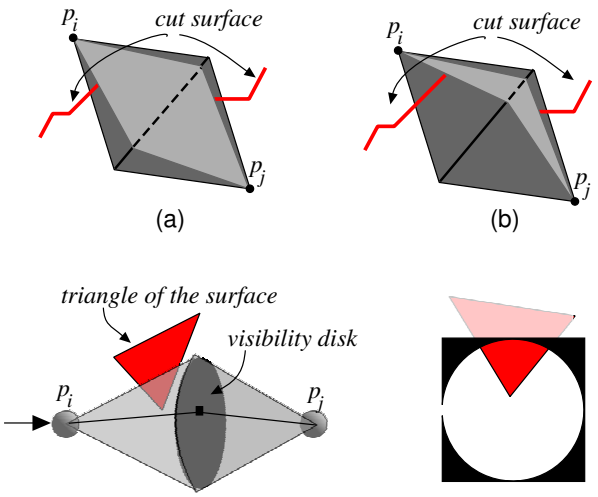


Fig. 11 (a-b) The cut surface partially occludes the visibility disk. The percentage of weight remaining is represented with a dashed line. (c) Hardware implementation of the Extended Visibility criterion. (d) a triangle of the cut surface as seen by phyxel i.

Taylor series. In other words the frame $f_{b'}$ is *virtual*, a concept already exploited by Molino et al. in their virtual node algorithm [14].

4 Extended Visibility criterion for physical response to cutting in MMs.

In the previous section we introduced the Splitting Cubes algorithm, which enables to introduce cuts and to setup the corresponding discontinuity in the deformation function F *inside* each single cell, without making any assumption on the physical model.
However, the deformation function must be changed to reflect the discontinuity introduced. This sections shows how to modify the deformation function to show discontinuities in MMs.

4.1 Nodes ¡-¿ Phyxels bounds

The deformation function at a grid node depends on the set of phyxels closer than a given radius r and that are visible, i.e. such that a segment from the node to the phyxel does not intersect the boundary surface. This set is referred as *kernel* of the node.
The kernel of a node is kept up to date when new pieces of surface are added. Updating a kernel means to check which of the phyxels in its kernel are still visible, so this operation requires few segment-triangle intersection tests only for the nodes within a radius r from the cell where the new piece of surface was created.

4.2 Phyxels ¡-¿ Phyxels bounds: the Extended Visibility Criterion

In Section 2 we reviewed the methods to update inter-phyxels relations after a cut. Here we propose to modify the visibility criterion by extending the line of sight between two phyxels from a segment to a pair of cones as shown in Figure 11.
Consider the common base of the two cones, that we call *visibility disk*. From each point on this disk we trace two rays, directed to each phyxel and say that this single point is *occluded* if at least one of these two rays intersects the cut surface. Then we define the weight function as:

$$w'(p_i, p_j) = w(p_i, p_j) \left(1 - \frac{1}{DiskArea} \int_{p \in Disk} IsOccluded(p) dp\right) \quad (3)$$

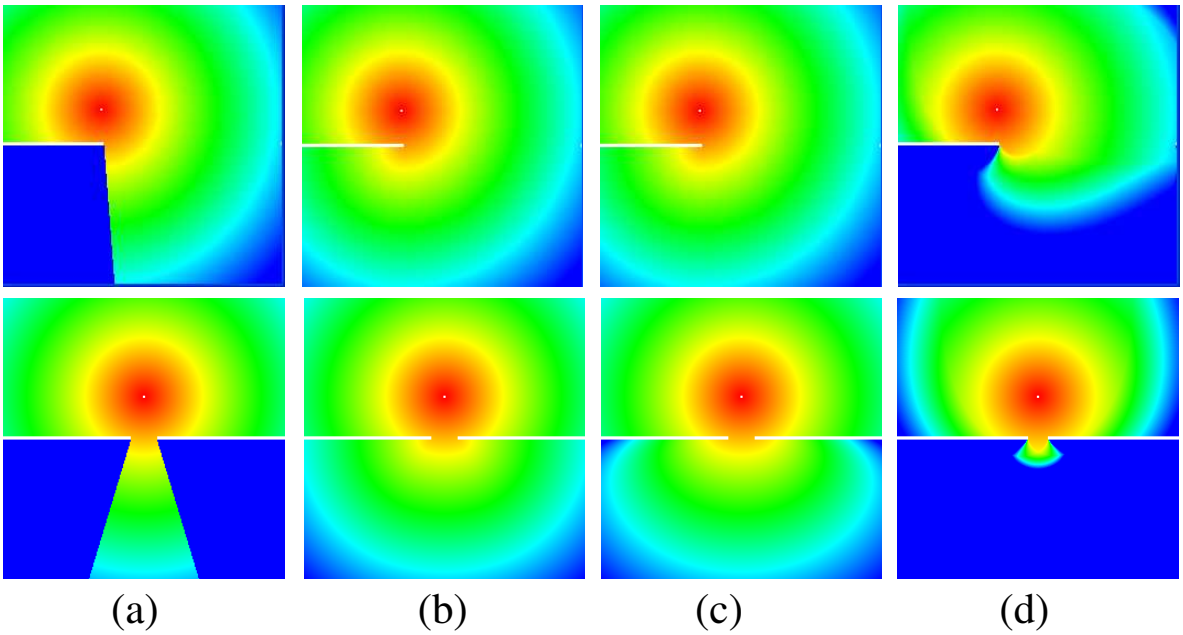


Fig. 12 The image shows how the weight function is modified by the introduction of a cut surface (white line) by using: the visibility criterion, the diffraction methods, the transparency method and the Extended Visibility criterion.

In this manner we replaced the binary value of the previous visibility criterion with a criterion which returns a scalar value in the range $[0, 1]$ that we use to weight the inter phyxel bound.
Figure 12 shows the value of the weight (in a color ramp from red= 1 to blue= 0) around a phyxel in the proximity of a cut surface when the visibility criterion, the diffraction method, the transparency method and the Extended Visibility criterion are used.
The first row shows the case were the cut surface is made of a single connected component. While the visibility criterion introduces discontinuities not only on the cut surface but also on the horizon line, the diffraction method and the transparency method implement the required discontinuity and are smooth elsewhere. The Extended Visibility criterion produces an intermediate result: the weight function does not exhibit unwanted discontinuities as the visibility criterion does, but decays around the tip point of the cut surface faster than using diffraction and transparency methods.
The second row shows the case where the cut surface is defined by two connected components, e.g. in time immediately before two crack fronts merge. Using either diffraction or transparency methods, the weight functions change discontinuously with respect to the growth of the cut surface at the point when the two cut surface merge, because both methods depend on the tip point. On the contrary, using the Extended Visibility criterion the weight function smoothly decays to 0 in the region under the cut surface. Note that the merging of crack fronts is a common event if we propagate cracks or if we perform a cut with scissors.

4.3 Implementing the Extended Visibility Criterion.

The choice of using cones may seems quite arbitrary but it is dictated by the possibility of implementing the Extended Visibility criterion entirely on the GPU. Let p_i and p_j be two *connected* phyxels, i.e. for which $w(p_i, p_j) \neq 0$, and cs a cut surface potentially occluding the disk between p_i and p_j. Consider the smallest square enclosing the occlusion disk. We associate a small single-channel texture to the square, and therefore to the visibility disk, which stores which samples of the disk have been occluded. This texture is permanently associated with the couple of phyxels and updated every time a new piece of surface (e.g. produced by the Splitting Cubes algorithm) could potentially occlude their visibility disk. Therefore we use $\#phyxels \cdot k$ small textures, where k is the average number of neighbors of a phyxels.

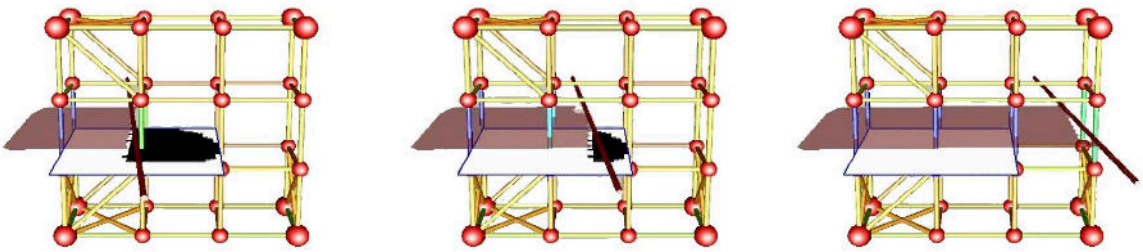


Fig. 13 A sequence of steps during a cut: the kernel value smoothly decreases as the visibility disc is 'obscured'.

To update the visibility disk we render *cs* twice: once from p_i towards p_j and once from p_j towards p_i, always setting the far plane of the projection on the mid point of the segment p_i and p_j and setting the projection so that each sample in the disk project on the same pixel for the two renderings. Using a fragment shader, we discard those fragments projecting on a pixel already written, so that all the fragments that are written into the frame buffer correspond to newly rasterized fragments, i.e. newly occluded samples. By using the hardware occlusion query, we count these fragments and update the weight as follows:

$$w'(p_i, p_j) = w(p_i, p_j)\frac{n - \#\dot{occluded}}{n} \tag{4}$$

which is the discrete version of equation 3 where n is the total size of the visibility disk in texels. The size of the texture used in this rendering step influences both smoothness of the weight function and performance. A large texture requires a longer rasterization time but provides a smoother change of the weight and viceversa. We found experimentally that a good tradeoff between smoothness and performance is assigning a radius of the transparency mask equal to half of the distance between the two phyxels and to use 32 buffers for rasterization (which means 16 pixel radii for the visibility disks).

5 Results

The approach presented in this paper was implemented on a Windows XP platform using C++ and OpenGL; the GPU code was written in GLSL. The tests shown in the accompanying video were all recorded in real time and have been run on a dual core P4 @ 3GHz PC equipped 2GB Ram, two HD 160GB SATA and a NVIDIA GeForce 8800GTX with 768MB. The initial surface is obtained by loading a watertight mesh and considering it as a big cut surface. Phyxels are regularly sampled in the volume. Once the cut is done, we obtain two surfaces: one bounding the object and one bounding the empty space around the object, that we simply throw away.

5.1 Efficiency and memory occupation

The graph in Figure 14 shows the performance of our approach recorded during the sequence titled **Liver** in the accompanying video. The object was filled with 566 phyxels and the initial surface created was made of 8452 triangles in 1177 grid cells.

The visibility disks where set to 16 pixel radius. Although the number of disks to update obviously depends on the speed to which the cut surface grows, the time spent in updating the visibility disks is almost constant during the cutting action. This is due to a time-critical implementation that assigns a fixed time slot per frame for disks updating, giving the possiblity to distribute the load on few consecutive frames. Note that the time for updating a disk is short and predictable (two renderings) and therefore the updating of all the disks is easily made interruptible by updating a few disks and then returning. It is clear that this may possibly causes a delay of the time step the cuts appears open but processing the disks in a FIFO order and distributing the load uniformly over the frames substantially avoids noticeable discontinuities during the opening of the cuts.

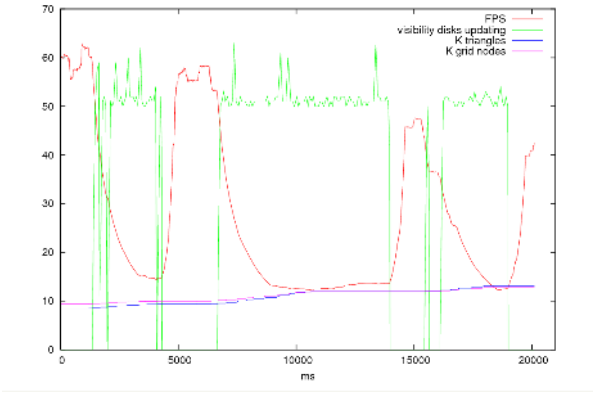


Fig. 14 Performance of the method recorder during sequence Liver

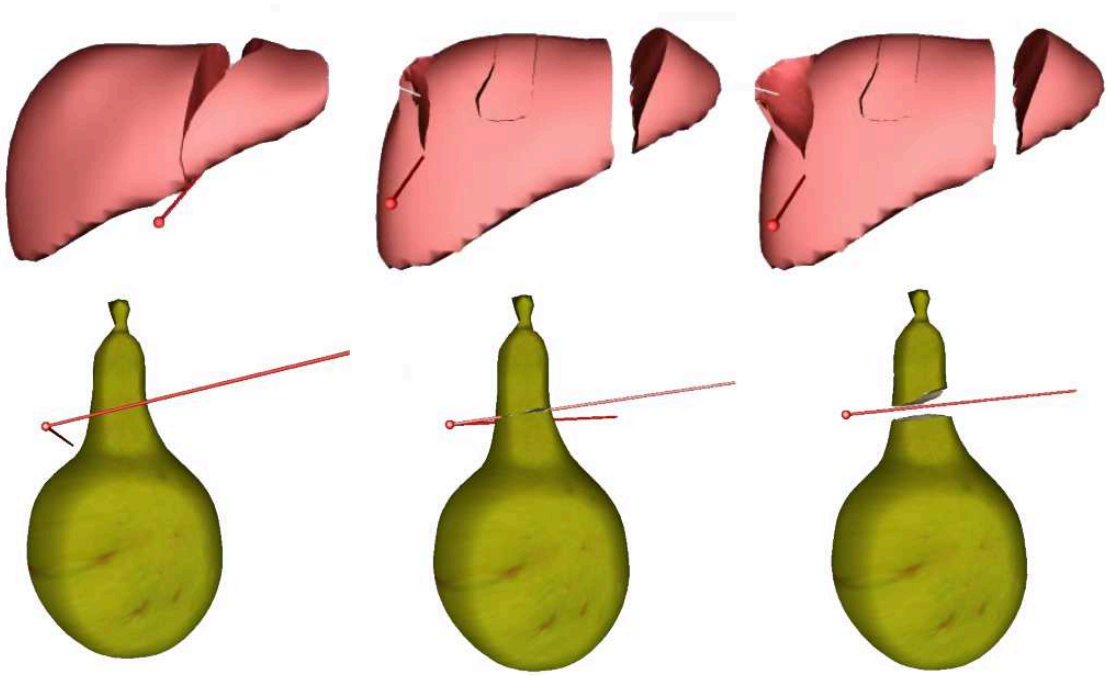


Fig. 15 Few frames from the accompanying video. Upper row: cutting of a model of the liver. Lower: scissors cutting a deformable pear.

The time spent to interpolate the tessellation vertices is 2ms for 4103 vertices. If we tessellate the surface by building a triangle mesh directly on the grid nodes, such a triangulation will have average triangle size of the order of a grid cell. In this sense, the time interpolation can be considered as an overhead introduced by the Splitting Cubes.
The time for generating the tessellation includes, for each grid cell: computing the cell configuration, accessing the LUT, instantiating new triangles and setting the dependencies of the vertices. The generation of the initial surface of the liver required to process 1177 cells an took 282 ms. In general, the processing of a grid cell takes less than 5 ms.

The memory occupation is also linearly related to the area of the cut surface (please note that only the grid nodes of the cells containing the surface are actually stored in main memory).
As expected, the number of triangles grows linearly with the cut surface.

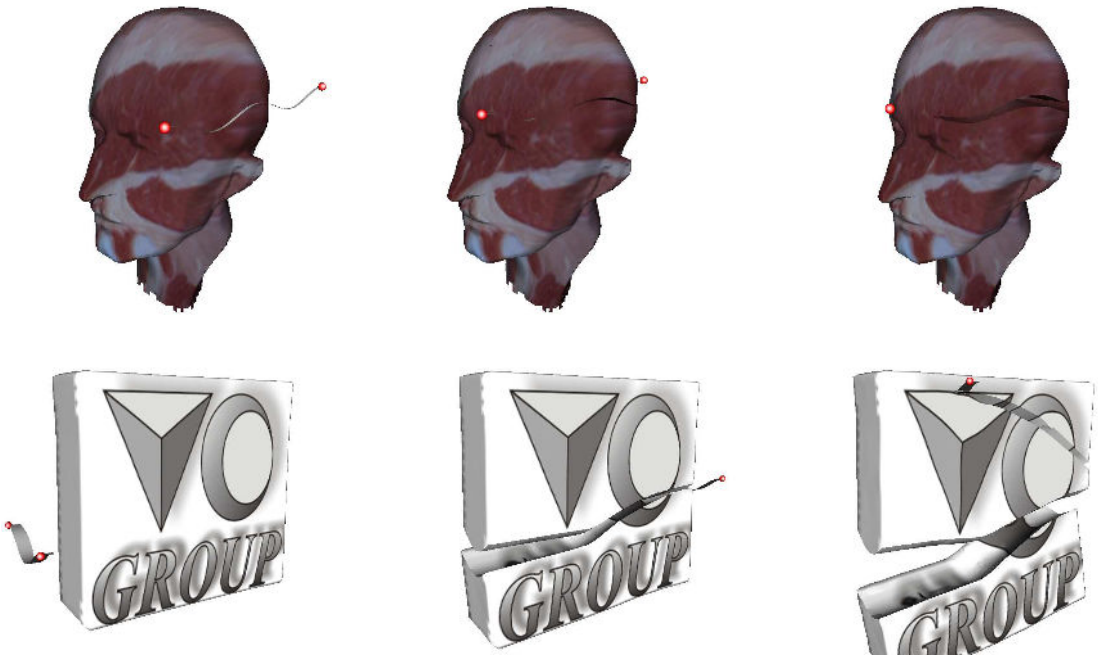


Fig. 16 Few frames from the accompanying video. In this case a curved tool is used.

5.2 Robustness and accuracy

The Splitting Cubes algorithm does not pose restrictions on the tool path. Thanks to the implicit definition of the tessellation, the surface is always intrinsically consistent and the self intersection of the cut surface does not need any special treatment.To support this claim one of the demos shows two cutting tools that act as scissors for cutting a pear while it deforms. The next sequence, titled **revolving blades** show a pair of parallel blades rototranslating into a cube-like object.

The video also shows a comparison between the visibility criterion and the Extended Visibility criterion (few frames are shown in Figure 13). We rendered the link between phyxels as a segment colored from red (completely visible) to blue (completely occluded) as in Figure 12 and the occluding surface (created during the cut). It can be seen how the cut performed using the Extended Visibility criterion is smoother. In the same video the weight functions are also compared in the case of merging cut surface.

5.3 Discussion

With respect to previous attempts towards this goal, the Splitting Cubes algorithm ehxibits highly desirable properties. First of all it is robust: since every possible configuration of cut edges corresponds to a predefined tessellation, there are no wrong states of the system that can lead the algorithm to end inconsistently. Furthermore, the task of detecting intersections betweeen the tool and the object is better conditioned than in mesh-based approaches, since we do not have to care about bad shaped triangles but only with the edges of a deformed regular grid. Another advantage is that our technique handles implicitly the changes of topology due to the cuts without having to analize the crack front as in [20,25]. Finally, the Splitting Cubes algorithm is conceptually simple, although not trivial to implement and, also thanks to the use of a LUT, efficient. However, the major benefit of the Splitting Cubes algorithm is that it decouples the model for physical simulation from the problem of virtual cutting (i.e. dynamic re-tessellation, intersection with other objects and local discontinuity) and therefore it can be seamlessy integrated with other methods for hysical simulation.
The Extended Visibility criterion gives a GPU friendly way to implement discontinuities on MMs with-

out the need for exploring the graph of adjacences as in [25] or the cut surface as in [25] but simply performing two renderings of few triangles for each couple of adjacent phyxels neigborhood of a cut.

6 Conclusions

In this paper we showed how interactive virtual cutting can be applied to MMs. In contrast with the previous solutions to this problem, the Splitting Cubes handles implicitly all the changes of topology due to the cuts and it is conceptually simple and robust. There are several directions of improvement/exploitation of the Splitting Cubes approach.

Original Boundary
With the current solution the detail of the representation and the granularity of the cut are coupled, since they both depend on the cell size. We could preserve the original tessellation of the boundary if we handled the intersection between the tessellation created using the LUT and the existing tessellation. This would not increase the granularity of the cut, but it would prevent visual artifacts related to the snapping of the vertices when the configuration of a cell changes.

Granularity of the cut.
The efficiency and speed of the Splitting Cubes algorithm comes to the price of limiting the granularity of topological changes to the level of the grid nodes, as it was for the virtual node algorithm. This limit can be overcome by considering that the ideas presented in this paper can be also implemented in an adaptive fashion, therefore replacing the grid with a hierarchy. However it is not immediate to switch to a hierarchical approach an mantain robustness and efficiency. For example, if we allow recursive subdivision of cells (tipically $1:8$), we reintroduce fragmentation and the growing number of cells (which would depend on the depth of the hierarchy) will undoubtely affect the performance. From the point of view of detecting intersections with the grid edges, again we will end up with a slower procedure, since we lose the implicit spatial indexing of the regular grid.
An important factor that must be taken into account is the scenario where virtual cutting is operated. In the application that motivates this research, i.e. a virtual cutting for surgical training, the interaction happens within a scale frame and therefore the limitation of granularity is unseen while the robustness and interactivity is a critical issue.

Collision detection.
As it can be easily seen in the videos the current implementation only supports collision with the cutting tool. However, the Splitting Cubes does not add any particular issue regarding the collision detection problem and existent techniques can seamlessly be integrated in the framework. We observe that cutting leads to critical situations where large portions of the surface are created in a situation of close contact. In this regard, the parametrization of the volume given by the grid could offer some advantage since we know which cells contain surfaces in close contact.

References

1. Belytschko, T., Krongauz, Y., Fleming, M., Organ, D., Liu, W.: Smoothing and accelerated computations in the element free galerkin method. J. Comput. Appl. Math. **74**(1-2), 111–126 (1996)
2. Belytschko, T., Lu, Y., Gu, L.: Element-free galerkin methods. Internat. J. Numer. Methods Engrg. (37), 229–256 (1994)
3. Bielser, D., Gross, M.: Interactive simulation of surgical cuts. In: Proceedings of the Pacific Graphics, pp. 116–125 (2000)
4. Bielser, D., Maiwald, V., Gross, M.: Interactive cuts through 3-dimensional soft tissue. Computer Graphics Forum (Eurographics'99 Proc.) **18**(3), C31–C38 (1999)
5. Bruyns, C., Senger, S.: Interactive cutting of 3D surface meshes. Computers & Graphics **25**(4), 635–642 (2001)
6. Cotin, H.D.S., Ayache, N.: A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In: CAS99 Proceedings, pp. 70–81 (1999)
7. Ganovelli, F., Cignoni, P., Montani, C., Scopigno, R.: A multiresolution model for soft objects supporting interactive cuts and lacerations. Computer Graphics Forum **19**(3) (2000)
8. Ganovelli, F., O'Sullivan, C.: Animating cuts with on-the-fly re-meshing. EuroGraphics Short Presentations, 2001. (J. C. Roberts, editor) (2001)

9. Garland, M.: Quadric-based polygonal surface simplification. Ph.D. thesis, Carnegie Mellon University, Computer Science Department (1999)
10. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring of hermite data. In: Siggraph 2002, Computer Graphics Proceedings, pp. 339–346. ACM Press / ACM SIGGRAPH / Addison Wesley Longman (2002). URL citeseer.ist.psu.edu/ju02dual.html
11. Kobbelt, L., Botsch, M., Schwanecke, U., Seidel, H.: Feature-sensitive surface extraction from volume data. In: E. Fiume (ed.) SIGGRAPH 2001, Computer Graphics Proceedings, pp. 57–66. ACM Press / ACM SIGGRAPH (2001)
12. Lim, Y.J., De, S.: On the use of meshfree methods and a geometry based surgical cutting in multimodal medical simulations. In: HAPTICS, pp. 295–301. IEEE Computer Society (2004)
13. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3D surface construction algorithm. In: ACM Computer Graphics (SIGGRAPH 87 Proceedings), vol. 21, pp. 163–170 (1987)
14. Molino, N., Bao, Z., Fedkiw, R.: A virtual node algorithm for changing mesh topology during simulation. ACM Trans. Graph **23**(3), 385–392 (2004)
15. Müller, M., Keiser, R., Nealen, A., Pauly, M., Gross, M., Alexa, M.: Point based animation of elastic, plastic and melting objects. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation (2004)
16. Nienhuy, H.W.: Cutting in deformable objects. Tech. rep., PhD Thesis, Utrecht University (2003)
17. O'Brien, J., A.W.Bargteil, Hodgins, J.: Graphical modeling and animation of ductile fracture. In: Proceedings of SIGGRAPH, pp. 291–294 (2002)
18. O'Brien, J.F., Hodgins, J.K.: Graphical modeling and animation of brittle fracture. In: SIGGRAPH, pp. 137–146 (1999)
19. Organ, D., Fleming, M., Terry, T., Belytschko, T.: Continuous meshless approximations for nonconvex bodies by diffraction and transparency. Computational Mechanics (18), 225–235 (1996)
20. Pauly, M., Keiser, R., Adams, B., Dutr;, P., Gross, M., Guibas, L.J.: Meshless animation of fracturing solids. ACM Trans. Graph. **24**(3), 957–964 (2005)
21. Pauly, M., Keiser, R., Kobbelt, L.P., Gross, M.: Shape modeling with point-sampled geometry. ACM Trans. Graph. **22**(3), 641–650 (2003). DOI http://doi.acm.org/10.1145/882262.882319
22. Sifakis, E., Der, K.G., Fedkiw, R.: Arbitrary cutting of deformable tetrahedralized objects. In: M. Gleicher, D. Thalmann (eds.) Symposium on Computer Animation, pp. 73–80. Eurographics Association (2007)
23. S.Li, Liu, W.: Meshfree Particle Methods. Springer (2004)
24. Steinemann, D., Harders, M., Gross, M., Szekely, G.: Hybrid cutting of deformable solids. In: IEEE VR 2006. IEEE (2006)
25. Steinemann, D., Otaduy, M., Gross, M.: Fast arbitrary splitting of deforming objects. In: Eurographics/SIGGRAPH Symposium on Computer Animation (2006)
26. Tanaka, A., Hirota, K., Kaneko, T.: Virtual cutting with force feedback. In: VRAIS '98: Proceedings of the Virtual Reality Annual International Symposium, p. 71. IEEE Computer Society, Washington, DC, USA (1998)
27. VisualComputingLab: Idolib: Interactive deformable objects library. Publicly available on web: `http://idolib.sf.net` (2005)
28. Wyvill, G., McPheeters, C., Wyvill, B.: Data structures for soft objects. The Visual Computer **2**(4), 227–234 (1986)

N. Pietroni received an advanced degree in Computer Science (Laurea) from the University of Pisa in 2003 and he is prensently a PHD student at university of Genova. He works at the Istituto di Scienza e Tecnologie dell'Informazione (ISTI) of the National Research Council (CNR) in Pisa, Italy. His research interests include modelling of deformable objects and texture systhesis.

F. Ganovelli is a research scientist with the Istituto di Scienza e Tecnologie dell'Informazione (ISTI) of the National Research Council (CNR) in Pisa, Italy. He received a PhD in Computer Science from the University of Pisa in 2001. His research interests include deformable objects modelling, collision detection, multiresolution and isosurfaces extraction.

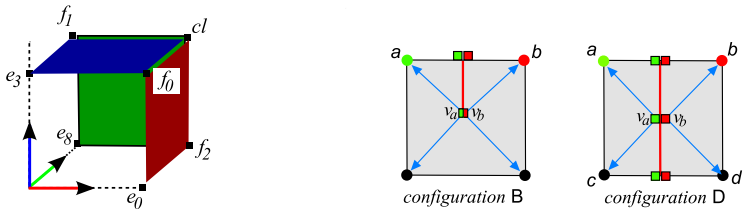
P. Cignoni is a Senior Research Scientist with ISTI-CNR. He received a Ph.D. Degree in Computer Science at the University of Pisa in 1998. He has been awarded "Best Young Researcher" by the Eurographics association in 2004. His research interests cover Computer Graphics fields ranging from visualization and processing of huge 3D datasets, to 3D scanning in the cultural heritage field and to Scientific Visualization. He has published more than ninety papers in international refereed journals/conferences.

R. Scopigno is a Research Director with ISTI-CNR and co-leads the Visual Computing Lab. He graduated in Computer Science at the University of Pisa in 1984. He is engaged in research projects concerned with 3D scanning, surface reconstruction, multiresolution data modeling and rendering, scientific visualization, volume rendering, and applications to cultural heritage. He published more than hundred twenty papers in international refereed journals/conferences and gave invited lectures or courses on visualization and graphics at international conferences. Roberto has been responsible person for ISTI-CNR in several EU projects. He was Co-Chair of international conferences (Eurographics '99, Rendering Symposium 2002, WSCG 2004, Geometry Processing Symp. 2004 and 2006, VAST 2005, Eurographics 2008) and serves in the programme committees of several events. He is Vice-Chair of the Eurographics Association, Co-Editor in Chief of the journal Computer Graphics Forum and member of the Editorial Board of ACM Computers & Cultural Heritage Journal.

7 Appendix A: Construction of the Look-Up-Table.

As explained in Section 3, each one of the 2^{12} possible configurations of the cut edges of a cell corresponds to a row of the LUT, which encodes two data: the tessellation representing the piece of surface internal to the cell and, for each vertex of such tessellation, its dependency from the cell nodes.

The construction of the LUT requires an algorithm that, given a configuration of cut edges, computes a tessellation for the cell. Fortunately we can take advantage from noticing that the tessellation of a cell can be expressed as the union of 8 tessellations, one for each cell node, and that each tessellation only depends on the cell edges entering the node. This can be better understood recalling the dual interpretation of the Splitting Cubes shown in Figure 6.(d) where the material (and therefore its boundary and the tessellation of its boundary) is associated to the grid nodes. We define the tessellation for a single node and then obtain the tessellation for the cell by rototranslating the vertices appropriately. Let us consider the tessellation for the node 0. For each cut edge leaving node 0 ($e0$, $e3$



(a) (b)

Fig. 17 (a) tessellation for the cell node 0. (b) two configurations that show why dependency canot be computed locally to the nodes.

and $e8$ respectively), we build a quad which has have one point on the edge, one point on each of the two faces sharing the edge and the point at the cube center (Figure 17.a). We obtain the tessellations for the other 7 nodes simply rototranslating the frames and reapplying this scheme. You may note that we will create duplicate vertices. As explained below, some vertices actually needs to be duplicated and some other does not; to know it which ones have to be duplicated we must compute their dependency.

Unlike the tessellation, the dependencies of the vertices cannot be treated locally to a cell node. Figure 17.b shows two configurations for a face. The vertices v_a and v_b are created both when tessellating node a and node b. In the configuration **B** the face is partially cut and v_a and v_b depend on the same set of grid nodes and therefore will always occupy the same spatial location. Conversely, in the configuration **D** v_a and v_b depends on two different set of nodes and therefore they will take apart.

As explained in Section 3, each vertex depends on the cell nodes of the same connected component. Therefore we applied a simple principle: a vertex depends on all the nodes of the same cell that are *reachable* through the material. We consider each grid node reachable by the vertices of its associated tessellation and two nodes of a cell as mutually reachable iff they are connected by a path of uncut edges.

Furthermore, since we need continuity of the deformation function across the cell boundary, the interpolation weights of the cell nodes for a edge-vertex will be 0 except for the 2 nodes of the edge and the interpolation weights of the cell nodes for a face-vertex will be 0 except for the 4 nodes of the face. Therefore we only need to find reachable nodes of the edge for the case of the edge-vertex (which is only one node) and reachable nodes of the face for the case face-vertex, since the weight for the other nodes would be zero even if they were reachable.

Back the tessellation, when two vertices have the same dependencies we do not instance two vertices, but we can statically unify them in the tessellation contained in the LUT so no useless duplicate vertex are created.

The LUT and the necessary code to read it can be found in the IDOLib library [27].