

CUTTING SIMULATION OF MANIFOLD VOLUMETRIC MESHES

C. FOREST¹, H. DELINGETTE¹ AND N. AYACHE¹

Abstract. One of the most difficult problem in surgical simulation consists in simulating the removal of soft tissue. This paper proposes an efficient method for locally refining and removing tetrahedra in a real-time surgical simulator. One of the key feature of this algorithm is that the tetrahedral mesh remains a 3-manifold volume during the cutting simulation. Furthermore, our approach minimizes the number of generated tetrahedra while trying to optimize their shape quality. The removal of tetrahedra is performed with a strategy which combines local refinement with the removal of neighboring tetrahedra when a topological singularity is found.

Résumé. La découpe volumique est l'une des principales difficultés en simulation de chirurgie. Cet article propose une méthode permettant le raffinement local et le retrait de tétraèdres dans le cadre d'un simulateur de chirurgie temps-réel. L'un des intérêts de l'algorithme est qu'il garantit la préservation du caractère *manifold* du maillage. De plus, notre approche minimise le nombre de tétraèdres créés tout en cherchant d'améliorer la qualité de ceux-ci. Le retrait de tétraèdres combine un raffinement local avec éventuellement le retrait de tétraèdres voisins lorsqu'une singularité est détectée.

INTRODUCTION

The simulation of cutting soft tissue is one of the major components of a surgical simulator. In fact, in a surgical simulation procedure, the word *cutting* may be used to describe two different actions: incising which can be performed with a scalpel, and removing soft tissue materials which is performed with an ultrasound cautery. Despite their different nature, the simulations of these two actions rise against a common problem, which is the topology modification of a volumetric mesh.

Incising algorithms have been the most commonly studied approaches in the literature. Most of them are based on subdivision algorithms whose principle is to divide each tetrahedron across a virtual surface defined by the path of the edge of a cutting tool [BG00, MK00]. These algorithms create a smooth and accurate surface of cut, but they suffer from two limitations. First, they tend to generate a large number of small tetrahedra of low shape quality. The number of new tetrahedra may be reduced by moving mesh vertices along the surface of cut [NvdS01], but this tends to worsen the quality of tetrahedra. Second, the simulation of incising supposes that the cut surface generated by the motion of a surgical tool is very smooth or even, as found in some articles, locally planar. However, in most real-time surgical simulation systems, there is no constraints on the motion of surgical tools and most of the time the cut surface is not even close to a smooth surface. Furthermore, surgical cutting gestures do not usually consist of a single large gesture, but are made of repeated small incisions. Therefore, these algorithms are not of practical use for cutting volumetric meshes, for instance when simulating an hepatectomy.

¹ Epidaure Research Project, INRIA Sophia Antipolis, 2004 route des Lucioles, 06902 Sophia Antipolis, France

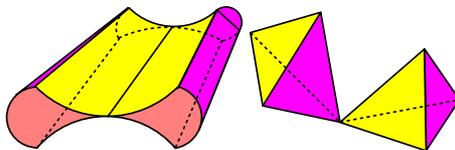


FIGURE 1. Examples of non manifold objects : (left) edge singularity; (right) vertex singularity

In this paper, we focus on the simulation of the removal of soft tissue material as performed with an ultrasound cautery. The targeted application is the simulation of hepatectomy, i.e. the resection of a functional segment of the liver. We suppose that all volumetric anatomical structures (for instance a liver) are represented with tetrahedral meshes which meet some criteria (see section 1.1). To remove soft tissue, we need to perform two distinct tasks : remove tetrahedral elements and control the element size such that the material cavities have the same size as the cautery device. The former task may seem trivial since it could reduce itself to a self-evident “**remove this tetrahedron from the list of tetrahedra**”. However, due to additional constraints on the mesh topology (which has to remain a manifold), it actually turns out to be a difficult problem to solve. The latter task consists in locally refining the mesh around the cut path. Indeed, in order to speed-up the deformation of soft tissue, it is preferable to use meshes with as few tetrahedra as possible. To obtain a realistic cut, it is therefore necessary to refine the mesh dynamically, as opposed to previous approaches [CDA00,PDA01] where the mesh was refined beforehand in regions where the cutting could occur.

In our approach, the tasks of removing and refining tetrahedra are both devised in order to optimize two constraints. First, they must minimize computation time and therefore cannot use sophisticated remeshing algorithm because each cutting operation should be performed in a few tens of milliseconds (typically 50-100 milliseconds). Second, they should produce tetrahedra with a high shape quality.

1. TETRAHEDRAL MESHES FOR SURGERY SIMULATION

1.1. Topological constraints

For the simulation of volumetric soft tissue, we use tetrahedral meshes as a geometric model. These meshes are built from triangulated surfaces with a dedicated mesh generation software. For our application, we chose to restrict the set of possible tetrahedral meshes to be both conformal and manifold. Conformality is required since we are using finite element modeling for the spatial discretization of the elastic energy (see [PDA01] for more details). In a conformal mesh, the intersection of two tetrahedra is either empty, or a common vertex, edge or triangle. Furthermore, the tetrahedral mesh is a 3-manifold [BY98] which implies that the neighborhood of a vertex is homeomorphic to a topological sphere for inner vertices and is homeomorphic to a half-sphere for surface vertices. More precisely, a tetrahedral mesh is a 3-manifold if the shell of a vertex (resp. an edge), i.e. the set of tetrahedra adjacent to a vertex (resp. an edge), has only one connected component. In Figure 1, we show two examples of non-manifold volumetric objects. When the neighborhood of a vertex or an edge is not singly-connected, we say that there exists a topological singularity at that vertex or edge. Having a manifold tetrahedral mesh is not mandatory for finite element algorithm. However because it allows to compute a normal for each surface vertex this feature is useful for the rendering of the mesh, for example when using Gouraud shading or PN triangles [VPBM01], and become necessary for computing the reaction force to be sent to a force-feedback device. Furthermore it simplifies the computation of edges and vertices neighborhood, allowing to decrease the redundancy of data structure.

Removing tetrahedra in conformal tetrahedral meshes is a trivial task but it proves to be much more difficult for manifold meshes. Thus, at least two authors have reported problems for removing topological singularities during cutting simulation [NvdS01,MLBdC01] without providing practical solutions.

1.2. Data Structure

We found the design of a data structure for a tetrahedral mesh to have a strong impact on the computational efficiency of the cutting simulation and on the ease of implementation. As a detailed description of this data structure would fall outside the scope of this paper, we will only describe its main features below.

Our data structure relies on the notion of manifold mesh and the topological information is mainly stored in vertices and tetrahedra. These two objects are stored in lists and each tetrahedron points towards its 4 vertices, 6 edges, 4 triangles and 4 neighboring tetrahedra. We also have edge (resp. triangle) objects which are stored in a hash table indexed by the reference of its two (resp. three) vertices. To each vertex (resp. edge), we add a pointer on a neighboring tetrahedron (resp. triangle) in order to build its neighborhood. We “close” topologically the volumetric mesh by adding virtual vertices, edges, triangles and tetrahedra for each surface vertex.

2. CUTTING ALGORITHM

2.1. Problem position

In our surgical simulator, the user manipulates a force-feedback device which has the same kinematics as a real surgical instrument in a laparoscopic procedure. The position and orientation of this virtual instrument is read periodically and collisions with the different virtual organs are detected with an efficient method [LCN99] based on standard OpenGL hardware acceleration. The input of the cutting algorithm is therefore a set of surface triangles and consequently a set of tetrahedra $\mathcal{T}_{initial}$ (adjacent to these triangles). When simulating an ultrasound cautery, all tetrahedra in the neighborhood of the tool extremity are simply removed.

We proceed in two stages : first neighboring tetrahedra are refined and then a subset of these tetrahedra are removed. We first describe the refinement stage before detailing the deletion stage.

2.2. Local Mesh Refinement

We have chosen, for the sake of efficiency and simplicity, to locally refine a mesh by splitting edges into two edges. More precisely, the input of the refinement algorithm is a set of edges : these edges may be inside or in the surface. Then, we compute the set of tetrahedra that are adjacent to any edges in the set. Finally, we split each tetrahedron in this set in a systematic manner which depends on the number and the adjacency of edges. There are 10 distinct configurations for refinement which are displayed in Figure 2. In order to get continuity across two neighboring tetrahedra (conformality), it is required in some configuration to take into account the lexicographic order of vertices (any other order between vertices could be used). When a single tetrahedron is split, we not only update the adjacency information between elements, but also the rendering information (texture coordinate) and the mechanical information.

The local refinement is controlled by a single distance threshold D_{ref} whose value depends on the diameter of the virtual tool (in our implementation $D_{ref} = 1.5 * \emptyset_{tool}$). Given the set of tetrahedra $\mathcal{T}_{initial}$ hit by the virtual tool, we build the set of edges \mathcal{E}_{split} to be split. For each tetrahedron in $\mathcal{T}_{initial}$, we scan each edge and add it into \mathcal{E}_{split} if its edge length is greater than D_{ref} . We do not use directly this set for the refinement because it would create elongated tetrahedra with a bad geometrical quality. Thus, we additionally scan each edge e in \mathcal{E}_{split} and look at all edges located in the shell of e (edges opposite to e inside each adjacent tetrahedron): if one of these edges has a length greater than p times the length of e then it is added to \mathcal{E}_{split} . The purpose of this extra stage is to limit the ratio between the minimum and maximum edge length for the newly created tetrahedra. Then the edge set is fed into the refinement procedure which produces as an output, the list of newly created tetrahedra \mathcal{T}_{new} . The refinement procedure is iterated until the edge set \mathcal{E}_{split} is empty, by replacing the set of initial tetrahedra $\mathcal{T}_{initial}$ with the list of new tetrahedra \mathcal{T}_{new} . The value p controls the mesh quality after refinement but also the extent of the refinement. If p is set to a value close to 1.0 then the refinement procedure may create a large set of new tetrahedra of high quality after many iterations. On the contrary, if p is set to a large value (2.0 for instance) then no extra refinement will take place and the refined tetrahedra

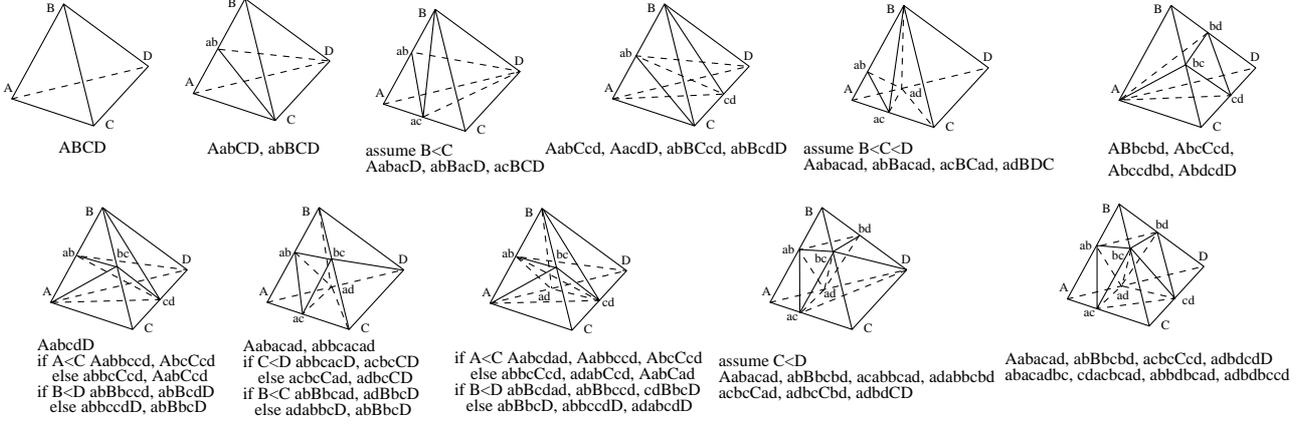


FIGURE 2. The 10 configurations for splitting a tetrahedron.

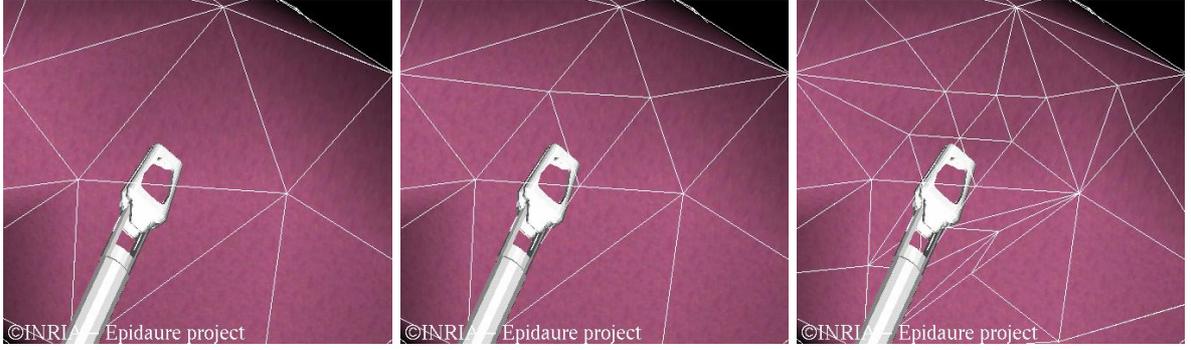


FIGURE 3. Iterative refinement of tetrahedra located at the vicinity of a virtual surgical tool

may be of poor quality. As a trade-off we are currently using $p = 1.5$ which, in practice, leads to no more than 3 refinement iterations.

2.3. Tetrahedra Removal

The second stage consists in removing tetrahedra. The list of initial tetrahedra intersected by the cautery device $\mathcal{T}_{initial}$ has been potentially extended by the refinement procedure. From this list, we discard any tetrahedron which is too far away from the tool or which has an edge with a length greater than D_{ref} and call $\mathcal{T}_{removal}$ the resulting set. The removal algorithm then consists in removing one-by-one all tetrahedra in $\mathcal{T}_{removal}$.

For a given tetrahedron T , we test if it can be removed without creating topological singularities at a vertex or edge. This is done according to the number of faces (triangles) of that tetrahedron T that are lying on the mesh surface :

Case 1 If T has no face in the mesh surface (T is then inside the mesh), it can be removed iff none of its four vertices belongs to the surface.

Case 2 If T has exactly one face belonging on the surface, it can be removed iff the vertex opposite to that face does not belong to the surface.

Case 3 If T has exactly two faces belonging to the surface, it can be removed iff the edge opposite to these two faces does not belong to the surface.

Case 4 If T has three or four faces belonging to the surface it can always be removed.

In fact, the probability that the suppression of a tetrahedron creates a topological singularity is fairly high. From a set of experiments described on section 2.6, we found (see Table 2.3) that this probability varies between 0.1 and 0.4 depending on the mesh topology or the mesh geometry. Also singularities at vertices (cases 1 and 2) occur nearly 10 times more often than singularities at edges (case 3).

Mesh	nb of operations	% of tetra removable	% of pbs. due to vertices	% of pbs. due to edges
1	130	89.3	9.2	1.5
2	265	62.3	31.7	6.0
3	460	84.1	13.5	2.4
4	1448	83.8	14.4	1.8
5	2268	86.1	13.3	0.6

FIGURE 4. Estimation of the occurrence of vertex or edge singularities when removing a tetrahedron on the mesh surface

There are different possible strategies for removing topological singularities. We have explored a variety of approaches and finally decided to use a combination of two strategies : the first one is based on a local mesh refinement and the second one is based on the removal of neighboring tetrahedra. All approaches were evaluated with respect to two criteria : the overall number and shape quality of tetrahedra. The local refinement strategy can always be used to solve topological singularities, but it tends to create many small tetrahedra (see section 2.4). Removing neighboring tetrahedra does not have this drawback but cannot be applied for all configurations (see section 2.5). We propose to combine the two previous strategies to get an optimal result (see section 2.7).

2.4. Basic strategy based on refinement

This strategy is quite simple : we create new tetrahedra surrounding the singularity in order to “thicken” the mesh at that location. If a topological singularity occurs at a vertex (resp. edge) then all edges adjacent to that vertex (resp. edge) are split (see section 2.2) and we then remove all refined tetrahedra adjacent to that vertex (resp. edge) together with the refined tetrahedra corresponding to the original tetrahedron. We can formally prove that these set of tetrahedra can always be removed without creating new singularities. This strategy works well in all cases but it creates many small tetrahedra of poor shape quality.

2.5. Strategy based on tetrahedra removal

Let T be the tetrahedron to be removed. The strategy consists in removing the smallest set of neighboring tetrahedra of T which can suppress a vertex or edge singularity without creating new ones. This is typically a search strategy however there is no guarantee that such set exists. This approach is well-suited for simulating surgical cuts since neighboring tetrahedra are also guaranteed to be small enough due to the local refinement stage. Furthermore, it is likely that neighboring tetrahedra of T also belongs to $\mathcal{T}_{removal}$ which implies that they would have been removed anyway. The algorithm significantly differs whether a vertex or an edge singularity is considered. For a vertex singularity, we can better understand the search problem by looking at the shell of the singular vertex.

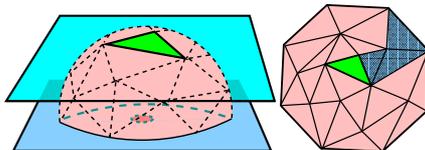


FIGURE 5 . (left)The shell of a vertex; (right) a path connecting tetrahedron T to the shell border

In Figure 2.5, we display the shell of that singular vertex: it is equivalent to a half-sphere since it lies on the surface. The opposite triangle of this vertex in tetrahedron T is drawn in solid line (leftmost figure) and dark grey (rightmost figure). To suppress the vertex singularity, we must find a set of adjacent tetrahedra (drawn as patterned triangles) that connect T to the border of the shell. The search is performed in a width-first manner in order to find the shortest possible path and is further arbitrarily limited to a depth of 10 in order to restrict the time spent in this search. When a path is found, its removability is tested and, when successful, the set $\mathcal{T}_{removable}$ is removed. If the test fails, we use an additional heuristics. It appears that the path we found, often divides the adjacency of one of the tetrahedron vertices into two connected components. Therefore, we also test the removability of the given path once extended with each of the two connected components. If one of those two sets appears to be actually removable it is temporally stored and will be eventually used if no smaller removable set is determined. If no paths nor extended paths are found to be removable then we output an empty set $\mathcal{T}_{removable} = \{\}$.

For an edge singularity, we look at all tetrahedra adjacent to that edge. Because this edge is lying on the surface, we can divide these tetrahedra into two sets, rightmost and leftmost tetrahedra. We first test each of these two sets for removal and in case of failure, we also test if the whole set of adjacent tetrahedra can be removed. If not, we output an empty set. This algorithm is very simple and can be implemented in a very efficient manner. We could also add more heuristics in case of failure to try to solve more configurations. However, edge singularities do not occur very often and therefore we have decided to keep it simple.

2.6. Strategy Efficiency

In order to estimate this algorithm efficiency, we used the electric lancet of our simulator to proceed a cut into several meshes.

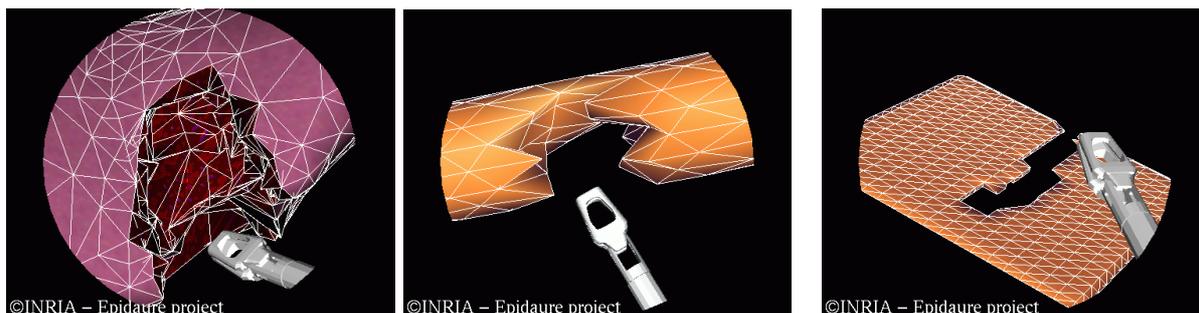


FIGURE 6. The meshes used for the tests

The meshes that were used are:

- (1) A simple cylinder
- (2) A thin sheet with an average thickness of two tetrahedra
- (3) A rough mesh of a liver (3970 tetrahedra) obtained with a 3D-Mesher

- (4) A locally refined mesh of the same liver (8454 tetrahedra), obtained with a 3D-Mesher
- (5) A locally refined of a liver (9933 tetrahedra for a third of the liver) obtained with a home-made edge splitting method

Table 1 shows the percentage of successful operations and the average cardinality of the removed set, for the whole cutting process, and restricted to both the subsets of tetrahedra that caused a problem at a vertex and at an edge. To be more significant, if a non-removable tetrahedra is encountered several times it is only counted once. The results appear to be quite stable from one mesh to the other and independent to the number of tetrahedra removed, to the geometry or the shape quality of the meshing. The only differences appear with the cylinder and the thin sheet, which is understandable because of their topological particularities.

Mesh	Vertices Singularities		Edges Singularities		Global	
	% of un-resolved	average cardinality	% of un-resolved	average cardinality	% of un-resolved	average cardinality
1	8.3	3.1	50.0	6.0	1.5	1.2
2	14.3	3.1	43.8	4.7	7.1	1.8
3	6.4	2.9	54.5	5.0	2.2	1.3
4	10.1	3.2	34.6	4.0	2.1	1.3
5	15.6	3.3	35.7	3.8	2.3	1.3

TABLE 1. Analysis of tetrahedra removal strategy

In average the strategy of removing neighboring tetrahedra allows to solve nearly 90 % of vertex singularities and 60 % of edge singularities. The total number of tetrahedra removed is nearly 3 for a vertex singularity and 4 for an edge singularity. Considering that vertex (resp. edge) singularities only occur 15 % (resp. 3 %) of the time when removing a tetrahedra, this strategy of removing neighboring tetrahedra allows to remove nearly 97 % of all tetrahedra that should be removed. Furthermore, on the whole, the average number of tetrahedra removed when a single tetrahedron should be removed is 1.3 which only shows a 30 % increase with respect to an ideal minimal value.

2.7. Combined Strategy : optimized local refinement

To remove the remaining 3 % of tetrahedra that cannot be removed by the previous strategy, we use an optimized local refinement strategy. Instead of refining (splitting) all edges adjacent to a vertex or edge singularity as described in section 2.4, we propose to refine only a subset of these edges. For instance, for a vertex singularity, it suffices to refine a set of tetrahedra that creates a path from T to the border of the vertex shell (see Figure 2.5). Thus, when searching for a suitable set of removable tetrahedra as described in the previous section, we also store the shortest path connecting the tetrahedron T to the shell border. This shortest path is eventually used for the optimized local refinement.

A similar approach is used for edge singularity where only a subset of adjacent edges are used corresponding to either the rightmost or leftmost set of adjacent tetrahedra.

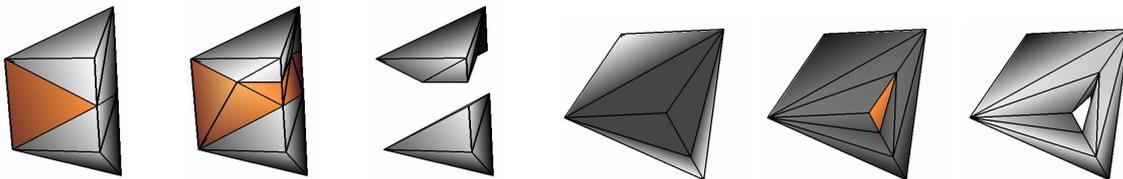


FIGURE 7. Removal of an edge(left) and of a vertex(right) singularity after an optimized local refinement

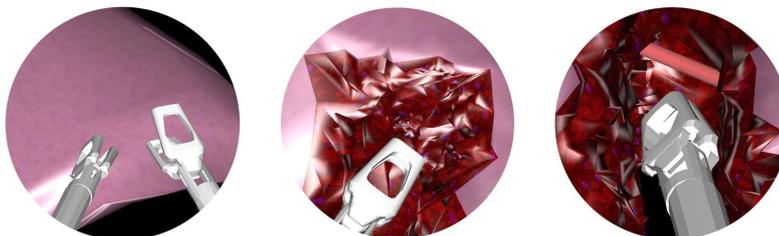


FIGURE 8. Sequence of a simulated liver resection with extricating of the portal vein

3. CONCLUSION

We proposed an algorithm which is suitable for the interactive simulation of the removal of soft tissue. The local refinement stage allows to simulate fine cuts even on a coarse tetrahedral mesh. The trade-off between mesh quality and computation efficiency is governed by a single parameter $p > 1$ whose value can be set intuitively. For removing tetrahedra, our approach combines the deletion of neighboring tetrahedra (in 97 % of cases) with the optimized refinement around topological singularities. A dedicated data structure for tetrahedral meshes has been an important component for its efficient implementation within a hepatic surgical simulator (see Figure 2.7).

REFERENCES

- [BG00] Daniel Bielser and Markus H. Gross. Interactive simulation of surgical cuts. In *Proc. Pacific Graphics 2000*, pages 116–125. IEEE Computer Society Press, October 2000.
- [BY98] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge University Press, UK, 1998. Translated by Hervé Brönnimann.
- [CDA00] S. Cotin, H. Delingette, and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. *The Visual Computer*, 16(8):437–452, 2000.
- [LCN99] J-C. Lombardo, M.P. Cani, and F. Neyret. Real-time collision detection for virtual surgery. In *Computer Animation*, Geneva Switzerland, May 26-28 1999.
- [MK00] Andrew B. Mor and Takeo Kanade. Modifying soft tissue models: Progressive cutting with minimal new element creation. In *MICCAI*, pages 598–607, 2000.
- [MLBdC01] C. Mendoza, C. Laugier, and F Boux de Casson. Virtual reality cutting phenomena using force feedback for surgery simulations. In *Proc. of the IMIVA workshop of MICCAI*, Utrecht(NL), 2001.
- [NvdS01] Han-Wen Nienhuys and A. Frank van der Stappen. Supporting cuts and finite element deformation in interactive surgery simulation. In W.J. Niessen and M.A. Viergever, editors, *Procs. of the Fourth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'01)*, pages 145–152. Springer Verlag, October 2001.
- [PDA01] G. Picinbono, H. Delingette, and N. Ayache. Non-linear and anisotropic elastic soft tissue models for medical simulation. In *ICRA2001: IEEE International Conference Robotics and Automation*, Seoul Korea, May 2001. Best conference paper award.
- [VPBM01] Alex Vlachos, Jorg Peters, Chas Boyd, and Jason L. Mitchell. Curved PN triangles. In *Symposium on Interactive 3D Graphics*, pages 159–166, 2001.