Volumetric modeling and interactive cutting of deformable bodies

Lenka Jerabkova, Guillaume Bousquet, Sébastien Barbier, François Faure, Jérémie Allard

Volumetric Modeling and Interactive Cutting of Deformable Bodies

Lenka Jeřábková[a], Guillaume Bousquet[a,b], Sébastien Barbier[a], François Faure[a,b], Jérémie Allard[c]

[a] *INRIA Rhône-Alpes & Laboratoire Jean Kuntzmann, France*
[b] *Université de Grenoble and CNRS, France*
[c] *INRIA Lille & LIFL, France*

Abstract

A new approach for the interactive simulation of viscoelastic object cutting is presented. Two synchronized geometrical models at different resolutions are used, both derived from medical images. In contrast with most previous approaches, the blade deforms the object, and cutting occurs once a contact pressure threshold is exceeded. Moreover, we achieve interactive simulation rates by embedding a high resolution geometry within a regular grid with arbitrary resolution. This allows to trade off accuracy for speed in the computation of deformations. The input data is a high-resolution volumetric model of the objects. The surface model of the object, used for rendering as well as collision detection and response, is a polygonal level set of the volumetric data. It is embedded in the volume model using barycentric coordinates.

Cutting is performed by removing voxels at the fine level, and updating the surface and volume models accordingly. We introduce a new data structure, which we call a Dynamic Branched Grid, in order to preserve the fine level topology at the coarse level. When an element of the coarse volumetric model is cut, it is replaced by a number of superimposed elements with the same size and at the same rest position as the original one. Each new element is assigned a part of material contained in the original one, and the mass and stiffness are recomputed accordingly. The well-known problem of creating small, ill-shaped finite elements while remeshing is thus completely avoided.

Keywords: biomechanics, soft tissue, cutting

1. Introduction

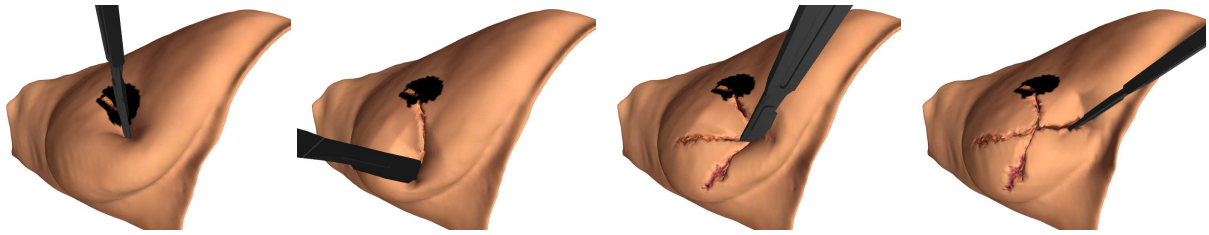
Although the simulation of cuts is essential for medical simulators, the interactive progressive cutting of deformable objects during the simulation is a challenging problem. The majority of current surgical simulators uses the finite element method (FEM) for tissue deformation. The number and quality of the finite elements have a direct impact on simulation performance and stability. Most methods for the modeling of cuts proposed so far require the finite elements to align with the cut. However, this often leads to severe stability problems caused by material slivers in the cut vicinity.

We propose a novel framework to optimize both the computational performance and the level of simulation fidelity perceived by the user, as shown in Figure 1. Different representations of the simulated object are used for different tasks such as visualization, collision detection, mechanical simulation and force feedback, as in Allard et al. (2007). A high-resolution textured surface model is used for the visualization while the mechanical simulation is based on a much coarser volumetric mesh, to provide a high level of visual details and interactive simulation response at the same time. In this paper, we tackle the challenge of dynamic topological changes in a patient-specific simulation.

1.1. Related Work

A FEM based simulation of tissue cutting requires modifications of the finite element (FE) mesh to represent the cut. This can be achieved by removing the elements touched by the cutting tool as in Cotin et al. (2000), by remeshing of the cut elements, see Bielser et al. (1999); Mor and Kanade (2000). While the former approach leads to unpleasing visual artifacts, the main drawback of the latter is the creation of small ill-shaped elements (slivers) causing numerical instability of the simulation. To avoid this drawback, Nienhuys

Email addresses: Lenka.Jerabkova@inrialpes.fr (Lenka Jeřábková), Guillaume.Bousquet@inrialpes.fr (Guillaume Bousquet)



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

Figure 1: This medically unrealistic example demonstrates the capabilities of our framework. (a) A scalpel pushes the surface of a patient-specific deformable model. (b) Cutting occurs when the pressure is sufficient. (c,d) The Dynamic Branched Grid easily handles multiple cuts.

and van der Stappen (2001) snap the nodes of the existing elements to the trajectory of the cut. Although no new element is created, this method can still lead to degenerated elements. Wicke et al. (2007) use an approach based on arbitrary convex finite elements. Although they remesh the FE mesh after a cut, the newly created elements are not subdivided into tetrahedra, thus avoiding the potential creation of many ill-conditioned elements. Slivers cannot be avoided completely, though.

In contrast to the previous methods, the virtual node algorithm by Molino et al. (2004) does not require the cut to be aligned with the elements' boundary. Instead of subdividing an element, one or more replicas of the cut element are created. The graphical representation of the material within the original element is fragmented, and the portions are assigned to the particular replicas. In order to avoid instabilities, the mass matrices of each element replica duplicate the material of the original element instead of splitting it along the crack. In the original virtual nodes algorithm the number of cuts per element is limited in that each fragment has to contain at least one node of the original FE mesh. This limitation was resolved by Sifakis et al. (2007), thus allowing for arbitrary cuts of tetrahedral elements. They do not mention though, how the masses of the fragments are updated. Moreover, they do not model the pressure applied by the cutting tool, resulting in "laser-cutting" effects.

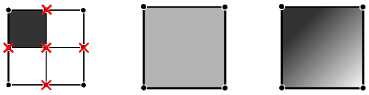
The extended finite elements method (XFEM) first proposed by Belytschko and Black (1999) for the simulation of cracks in structural mechanics can effectively model discontinuity regions within an FEM mesh by introducing a local enrichment in subregions with discontinuities. The XFEM does not require the alignment of the cut with the FE mesh and it has been used for surgical simulation in Jeřábková and Kuhlen (2009). However, it does not handle multiple cuts within a single element.

1.2. Contribution

From the viewpoint of dynamics simulation, our approach is closely related to the virtual node algorithm and the XFEM. However, in contrary to most methods, which use tetrahedral elements, we use a two level hierarchy of hexahedral elements. Due to the fixed size and well defined shape of the hexahedra, ill-shaped elements cannot occur, which dramatically simplifies the remeshing problem. Moreover, the similarity of shapes allows us to easily build a hierarchy of coarse voxels based on fine ones. This simplifies the construction of physical models based on volumetric images.

The mechanical simulation is based on coarse level finite elements, whereas the visualization and collision models use a fine level of detail. The proposed voxel-based approach is particularly suitable for medical simulation, because it can directly use 3D medical data, which defines the finest level of detail. Arbitrarily coarse levels of detail can be generated recursively by doubling the voxel size at each level. Using these nested grids, we avoid the problems related to mesh coupling that arise when trying to find a consistent mapping between different mesh-based representations. Special attention has to be paid when independent or branched parts of objects fall into a single coarse element. In section 2 we propose a solution which allows the propagation of fine topological structures into the coarse levels during the simulation, thus enabling the creation of interactive incisions. The partially empty voxels located on the boundaries of the objects or created by incisions are modeled using non-uniform stiffness as in Nesme et al. (2006), as explained in section 2.2. We extend their approach to the modeling of objects based on volumetric images, and model object surfaces using isosurfaces.

Furthermore, to our knowledge, no existing method models contact pressure while cutting. Our solution manages such a contact response, as presented in section 3. We model the cutting tool using two shapes



(a) (b) (c)

Figure 2: The principle of non-uniform stiffness. (a) fine level element, (b) coarse level element with uniform stiffness, (c) coarse level element with non-uniform stiffness.

embedded in each other. The outside shape is used to repel the object based on penalties using a fast GPU-based method. When it sufficiently penetrates the object, fine-level voxels are intersected by the inside shape and removed. The voxel hierarchy is then updated accordingly. This simulates both the deformation due to the pressure of the cutting tool and the dissection.

Our results and discussions are presented in section 4.

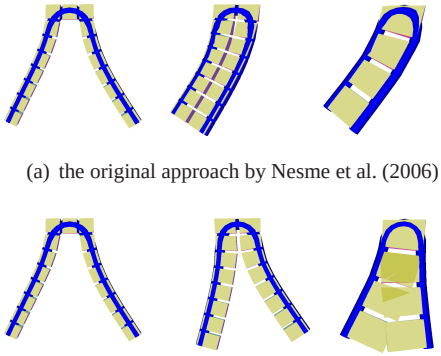
2. Dynamic Branched Grid

The volumetric model of the object, used to compute the deformations and to apply the viscoelastic response, is a regular grid of arbitrarily coarse deformable hexahedral elements, hierarchically built on top of fine voxels. In order to reflect the fact that the coarse voxels may be partially empty, a non-uniform stiffness and mass distribution is used as described in Nesme et al. (2006) and illustrated in Figure 2. This allows us to adjust a trade off between accuracy and speed of the dynamics while preserving a high visual quality of the simulation.

2.1. Topology Update

An important improvement of the original approach presented in Nesme et al. (2006) is the preservation of object topology at coarse levels, as illustrated in Figure 3.

At the finest level, it is assumed that each cell only contains one continuous piece of material. Based on this, a connectivity graph is created, with nodes representing the finite elements of the fine level connected by edges when linking material is present. For each cell of the coarse level, we determine the number of independent material components, using the connectivity graph from the previous finer level. One finite element is created for each independent, connected graph component within a cell. The number of superimposed elements per cell is not limited. While they all have the same extent as the cell they occupy, they are only partially filled with material. Since the collision detection uses the embedded material, no collision is detected between the



(a) the original approach by Nesme et al. (2006)

(b) Dynamic Branched Grid

Figure 3: The principle of branching. A deformable U-shaped tube (blue) is simulated using hexahedral elements (yellow) at different resolutions. The original method requires a fine resolution (left) in order to simulate both ends of the tube independently whereas the Dynamic Branched Grid connects the elements at coarser resolutions (center and right) according to the real material connectivity. At the coarsest resolution (right), our method creates superimposed elements.

superimposed elements at the beginning of the simulation. They can then move apart and eventually collide during the simulation.

Each element is connected to its adjacent elements according to material connectivity at the cell borders, and the topology-preserving coarsening process can continue upward.

We improve on Nesme et al. (2009) by extending the method to dynamic topologies to create incisions during the simulation (see Figure 4). Cutting is implemented as voxel removal on the fine level. After each voxel removal, the connectivity graph is locally updated and coarse voxels are reconnected if necessary. The following situations can arise at the coarse level:

- **cell removal**: if the removed fine voxels were the last voxels of a coarse element, the coarse element is removed.
- **cell split**: if the removal of fine voxels leads to the creation of one or more independent material components at the coarse level, new superimposed finite elements are created such that each of them contains one independent material component. The corresponding nodes are connected according to the material connectivity of the new elements.
- **cell disconnect**: if the removed fine voxels were the last voxels connecting a coarse element to one or more of its neighbors, these have to be mechanically disconnected.

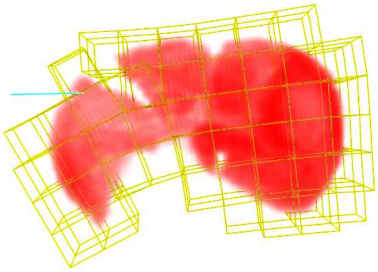
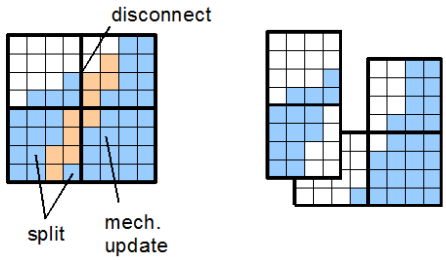


Figure 4: Branching in 3D. Fine level voxels of a segmented liver (red) are embedded into a branched grid of coarse finite elements (yellow). When a cut is created, branching is performed on the fly. Blue line depicts an interaction force.



(a) Four elements with an incision (orange).

(b) The resulting elements.

Figure 5: Topological changes resulting from an incision. The upper two elements were disconnected, whereas the lower left element has been split in order to represent two independent material components.

- **no topological change**: if the removal of fine voxels does not cause any connectivity change at the coarse level, the corresponding element remains topologically unchanged but its mass and its stiffness are updated.

The latter three cases are demonstrated in Figure 5.

2.2. Update of Non-Uniform Stiffness Elements

When a new element is created (e.g., due to the split operation as described in Sec. 2), its mechanical properties are initialized based on the material distribution inside the element. Depending on the coarsening level and the number of full fine voxels, this can be a time consuming operation when done recursively as proposed in Nesme et al. (2006). The method proposed in Nesme et al. (2009) uses static deformation analysis of the composite elements in order to compute resulting shape functions of the elements. This leads to better results than Nesme et al. (2006), but at a significantly higher precomputation and computation cost, incompatible with interactive applications. We propose an optimisation of Nesme et al. (2006) in order to directly update the mechanical properties at the coarse level from the modifications done at the fine level, allowing a faster update while handling topological changes.

In Nesme et al. (2006), the mass and stiffness matrices at coarse level are computed using recursive matrix condensation following an octree scheme ($8 \rightarrow 1$), as illustrated in Figure 7(a). Starting at the finest level, the contributions of all non-empty elements are added to the parent element based on the location of the element with respect to its parent:

$$A_{(l+1)} = A_{(l+1)} + H_i^T A_{(l)i} H_i \tag{1}$$

where H_i is the interpolation matrix from an i-th fine child element into a coarse element and A represents either a mass matrix M or a stiffness matrix K at a given level (the $A_{(0)i}$ being the matrices of the finest elements).

We propose a direct condensation method from any intermediate level to the coarse level ($n \rightarrow 1$), see Figure 7(b). We analyze the material distribution within the coarse element and detect full octree subnodes at all levels between the fine voxels and the coarse level element. The stiffness and mass matrices of the corresponding full elements at intermediate level l can be straightforwardly computed from the fine level matrices as:

$$K_{(l)} = 2^l K_{(0)} \tag{2}$$

$$M_{(l)} = 2^{3l} M_{(0)} \tag{3}$$

and added to the coarsest level L matrices using an interpolation matrix $H_{(L-l)i}$

$$A_{(L)} = A_{(L)} + H_{(L-l)i}^T A_{(l)} H_{(L-l)i} \tag{4}$$

$(L - l)$ is the level difference between the current octree subnode and the root node, the index i denotes the i-th child and can vary between $(0 - 2^{3(L-l)})$.

Similarly, when a fine element i is deleted, its contribution is subtracted from the coarse matrix:

$$A_{(L)} = A_{(L)} - H_{(L)i}^T A_{(0)} H_{(L)i} \tag{5}$$

It remains to explain how do we detect the full octree subnodes within a coarse element. Note that this step can be done efficiently without actually building an octree. Figure 6 illustrates the principle in 2D using a quadtree with 3 coarsening levels. When an octree specific voxel numbering is used, the i-th node at level l will correspond to an interval of 2^{3l} voxel indices starting with $i \cdot 2^{3l}$. This octree index, assigned to each voxel, is similar to Morton index, also known as Z-index. It can be obtained from the binary representation of its x, y, z

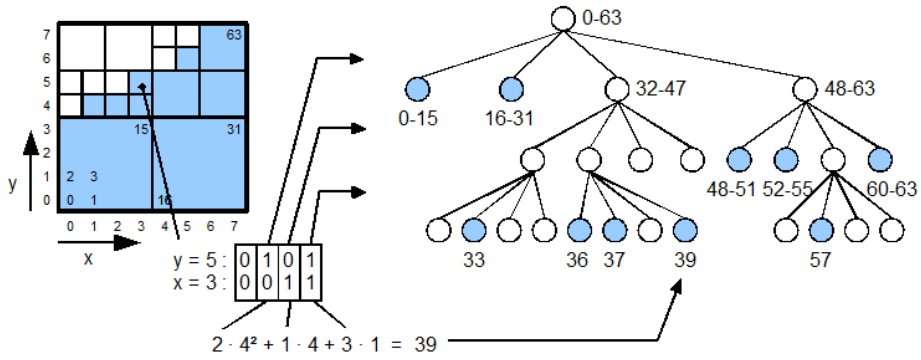
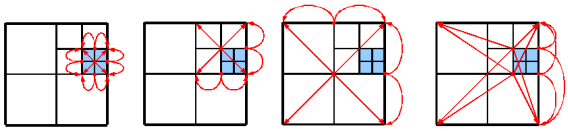


Figure 6: A partially filled coarse element at level 3 with its quadtree representation. Numbers within fine elements denote the corresponding quadtree indices. An example index computation is shown for node 39 ($x = 3, y = 5$).



(a) recursive update (b) direct update

Figure 7: Recursive versus direct update. In this example, four level 0 elements (blue) compose a full element at level 1. The red arrows represent contributions of the nodes of a fine element to the nodes of a coarser element. The recursive method computes the element contributions starting from the finest level through all the intermediate levels, whereas the direct method directly updates the coarsest level.

coordinates by taking the bit triplets in highest to lowest order and interpreting them as the index of octree node child to be chosen to store the voxel. The final voxel index in an octree will thus evaluate to

$$Idx_{octree} = \sum_{l=0}^{L} (4 \cdot z_l + 2 \cdot y_l + x_l) \cdot 8^l \qquad (6)$$

where x_l, y_l, z_l denote the l-th bit of the x, y, z coordinates respectively. Once we know the octree indices of all fine voxels, we have to detect intervals corresponding to full octree subnodes. For example, the coarse node in Figure 6 contains the following voxel intervals:

`[0-31] [33] [36-37] [39] [48-55] [57] [60-63]`

corresponding to the quadtree nodes

`[0-15] [16-31] [33] [36] [37] [39]`
`[48-51] [52-55] [57] [60-63]`

The contribution of each of these nodes is added to the coarse level matrices in one condensation step per node using Equation (4), thus resulting in 10 condensation steps. The original recursive method would require 49

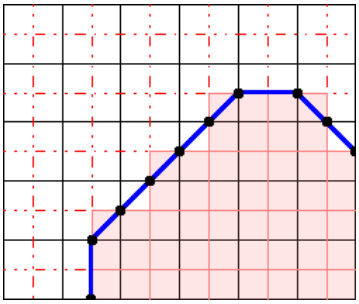


Figure 8: Voxels are in red, the marching cubes grid in black, the triangular surface in blue and the vertices as black points. Each center of voxel defines a corner of the marching cubes grid. It gives a direct access between triangles and hexahedra (representing voxels).

condensation steps at level 0, 14 steps at level 1 and finally 4 steps at level 2, i.e. a total of 67 condensation steps.

In the case of inhomogeneous materials, equation 5 for fast element update remains valid. However, for cell splitting, we can not accurately compute the matrices of inhomogeneous voxels using equations 2 and 3, and we would have to consider homogeneous voxels separately at the appropriate level. The efficiency of the improved update mechanism would depend on the way the materials are interleaved, e.g. two materials with a clear border between them would allow for a faster update than small parts of one material scattered inside another material. We defer this issue to future work.

3. Surface Model and Cutting

A surface is used for rendering and collision detection. We model it using a polygonal level set (Lorensen and Cline (1987)) based on the fine voxels, and embedded within the coarse deformable elements, as il-

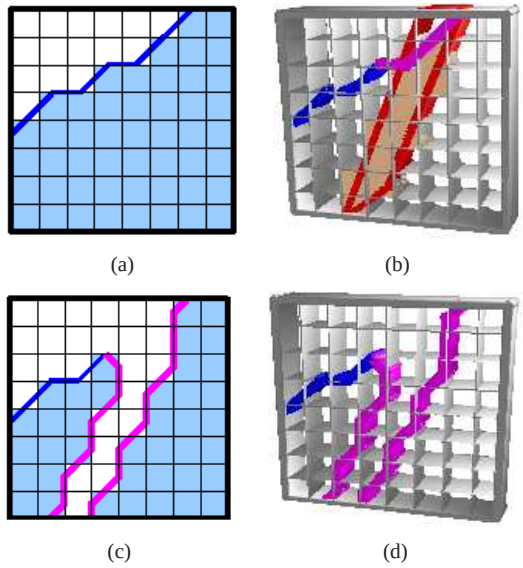
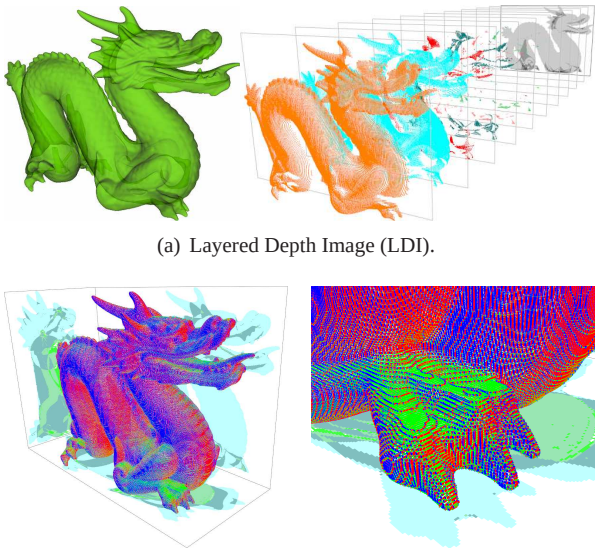


Figure 9: Embedded surface. The light blue squares denote the object voxels in the medical image. (a) Embedded level set surface (blue) in image space, (b) The deformed surface, intersected by a cutting blade (red). Voxels inside the intersection volume (orange) will be removed. (c) Surface update in image space, (d) The updated deformed surface.



(a) Layered Depth Image (LDI).

(b) Multi-axis LDI (left) and close-up view (right).

Figure 10: (a) A Layered Depth Image (LDI), represented here as a series of 2.5D images, stores the depth of all layers below each pixel. (b) Combining LDIs along the three spatial axis provides an accurate volumetric representation.

lustrated in Figure 9(a) . The vertices are attached to the deformable cells, shown in bold black, using their barycentric coordinates.

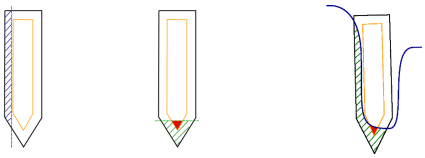
To connect triangles and voxels, each triangle stores its index in the marching cube regular grid at its creation. The center of the voxels define the intersections of this grid, as shown in the Figure 8. Thus, if we shift the voxel grid by the half size of a voxel, we see that we have the same indices for the voxels and the cubes in the marching cubes grid and we can access directly to the voxels defining a triangle.

To detect all the collisions and self-collisions during the simulation, we apply the GPU-based method proposed by Faure et al. (2008), which computes elastic forces applied to the triangular models, which are then mapped to the coarse level hexahedra and used to implement repulsion. Beside performance, the main strength of this approach is to robustly recover from volume intersection and self-intersection, without distance fields or any other precomputation. This allows us to let the simulation run while cutting, in contrast to many other methods. We extend this method using CUDA kernels on the GPU to not only rasterize the surfaces but also compute the intersections (see Figure 10). As this process is fully done on the GPU, the synchronization and transfer costs are significantly reduced and we achieve much higher performance. Note that other works have proposed even faster methods to rasterize LDIs (Liu et al. (2009)), however at the cost of missing layers when surfaces are very close to each other, which is a common case for collisions.

To model cutting, we extend the collision processing algorithm as follows. The collision between the deformed surface and the cutting tool is detected using the GPU, and the list of the triangles on the surface of the intersection volume, as shown in magenta in Figure 9(b), is returned to the CPU. These surface triangles allow us to get the list of the closest voxels. A propagation starts from these voxels, until all the voxels centers (interpolated based on the coarse voxels) inside the intersection volume are detected, as depicted in Figure 9(b). The found voxels are removed, and the level set surface is recomputed in the undeformed image, as illustrated in Figure 9(c).

The update of the level set surface is performed locally. Each removed voxel index in the regular grid gives a direct access to the triangles to remove, because each voxel defines a corner of a cube in the marching cube, as shown in Figure 8. These indices give also the minimal set of cubes in the marching cube to compute to obtain the new polyhedral patch, which is the set of cubes (partially) defined by the removed voxels.

Moreover, the branched grid is locally updated, using the method presented in section 2, which allows cut



(a) Two collisions with the same intersecting volume. The vertical one (in blue) just collides while the horizontal one (in green) cuts as the collision surfaces are different.

(b) The blade plate pushes while the sharp edge cuts the intersected model.

Figure 11: Cutting using nested models. The cutting model, in orange, is placed inside the repulsion model, in black.

opening, and ultimately object splitting. The embedding of the updated surface in the deformable cell is then applied, as illustrated in Figure 9(d).

The marching cubes algorithm often produces sharp edges meshes. We smooth the surfaces to obtain a better visual model. As each triangle stores its connection with the hexahedra, displacing the vertices of the triangles with a smoothing algorithm is not problematic for the other operations.

To implement a contact response while cutting, we only want to remove voxels when the pressure exceeds a given threshold. Our approach uses two nested geometrical models, one for repulsion and one for cutting, as illustrated in Figure 11, which shows different scenarios of collision of a cutting object with an organ. In the left, the contact with the blue object occurs on a large surface, thus for a given force the resulting penetration is not deep and no cutting occurs. In the middle, the contact with green object occurs on a smaller surface, thus the same force results in a deeper penetration, which allows an intersection between the cutting model (orange, inside) and the voxels. In the right, a single object is both repulsed and cut. An application of our model is shown in Figure 1.

We use the graphics pipeline to achieve the realistic rendering of the simulated models during the simulation. Each vertex of the extracted surface has a material identifier according to the segmented medical data. Different textures and rendering parameters are associated with materials such as skin, flesh or fat. For each triangle, barycentric coordinates are computed to obtain a correct interpolated color and lighting for each pixel. The texture coordinates are extracted from initial positions, normals and segmentation id of each vertex. Bump mapping can be activated if needed. This high quality rendering provides improved depth cues and allows a better object distinction during the interaction.

4. Results and Discussion

4.1. Performance

Figure 12 shows a sequence of screenshots of a liver surgery simulation. The liver consists of approx. 13.000 fine level voxels. Three coarsening levels are used, resulting in 104 elements for the mechanical simulation. This scene runs at 40 fps. The simulation setup enables bi-manual interaction using two Phantom haptic devices for grasping and cutting respectively. To achieve the necessary 1kHz update rate for smooth haptics, a coupling algorithm is used which extrapolates the missing collision updates.

The breast scene illustrated in Figure 1 is composed of 519 coarse voxels, 96702 fine voxels, 93532 triangles, and runs at 15.7 fps when pushing and cutting.

The time spent in each step heavily depends on the dimension of the rasterized box (defined by the intersection of the collision model boxes) and on the number of voxels to remove. In average, 60% of the time is spent in the PDE solution, the computation of the FEM forces and the integration of the positions. The collision detection performed on the GPU takes 15%, the determination of the voxels to remove by propagation 2%, the branched grid update 10%, and the triangular remeshing 2%.

Currently, an important performance issue comes from the CPU implementation of the embedding of the surface within the coarse hexahedra. The vertex coordinates are transmitted at each time step to the GPU. Moreover, to change topology, the GPU returns a list of fine voxels to remove, the CPU updates the Dynamic Branched Grid, recomputes the surface voxels and connectivity, and transfers them to the GPU for rendering and collision detection. The whole surface is transmitted each time, including the unmodified connectivity areas. A full GPU implementation of the model would dramatically increase the performance.

4.2. Limitations and Future Work

Our approach relies on fine-level voxels to model object surface and to simulate cutting. This volume data requires more memory space than traditional, surface-based models. Cutting is performed by removing voxels. While for sufficiently small voxels this typically remains unnoticeable, it may result in significant volume loss in case of a large number of cuts. Separating rather than removing voxels is an improvement to introduce in future work. Additionally, applying different mechanical properties to different kinds of tissue based on voxel color is a straightforward extension. Finally, while the GPU-based collision response method is very robust to

Figure 12: Liver surgery simulation. Left: rest state. Center and right: the liver is deformed and cut interactively.

intersections, the (un)stability of our hexahedral FEM does not yet allow us to simulate scenarios where the blade is heavily compressed between the two parts of the cut object.

4.3. Conclusion

The proposed dynamic branching grid enables one to build models for simulation based on volumetric images such as segmented medical data. The grid can be built automatically at any arbitrary resolution while respecting the topology of the simulated object. Using this structure, a new approach to handle interactive cutting has been demonstrated. Our method avoids the creation of ill-shape elements. Moreover, we have presented a new cutting tool model which pushes the object while cutting, providing more realism. In future work we will address the current limitations and performance issues in order to progress toward the realistic real-time cutting simulation of highly complex objects.

Allard, J., Cotin, S., Faure, F., Bensoussan, P.J., Poyer, F., Duriez, C., Delingette, H., Grisoni, L., 2007. SOFA an Open Source Framework for Medical Simulation, in: Medicine Meets Virtual Reality (MMVR'15), Long Beach, USA.

Belytschko, T., Black, T., 1999. Elastic crack growth in finite elements with minimal remeshing. International Journal for Numerical Methods in Engineering 45, 601–620.

Bielser, D., Maiwald, V.A., Gross, M.H., 1999. Interactive Cuts through 3-Dimensional Soft Tissue, in: Proceedings of the European Association for Computer Graphics 20th Annual Conference. Eurographics'99., pp. 31–38.

Cotin, S., Delingette, H., Ayache, N., 2000. A hybrid elastic model for real-time cutting, deformations and force feedback for surgery training and simulation. The Visual Computer 16, 437–452.

Faure, F., Barbier, S., Allard, J., Falipou, F., 2008. Image-based collision detection and response between arbitrary volumetric objects, in: ACM Siggraph/Eurographics Symposium on Computer Animation, SCA 2008, July, 2008, Dublin, Irland.

Jeřábková, L., Kuhlen, T., 2009. Stable cutting of deformable objects in virtual environments using the XFEM. IEEE Computer Graphics and Applications 29, 61–71.

Liu, F., Huang, M.C., Liu, X.H., Wu, E.H., 2009. Efficient depth peeling via bucket sort, in: HPG '09: Proceedings of the Conference on High Performance Graphics 2009, ACM. pp. 51–57.

Lorensen, W.E., Cline, H.E., 1987. Marching cubes: A high resolution 3d surface construction algorithm, in: SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA. pp. 163–169.

Molino, N., Bao, Z., Fedkiw, R., 2004. A virtual node algorithm for changing mesh topology during simulation. ACM Transactions on Graphics 23, 385–392. Special Issue: Proceedings of the 2004 SIGGRAPH Conference.

Mor, A.B., Kanade, T., 2000. Modifying Soft Tissue Models: Progressive Cutting with Minimal New Element Creation, in: Proceedings of Medical Image Computing & Computer Assisted Intervention, pp. 598–607.

Nesme, M., Kry, P., Jerabkova, L., Faure, F., 2009. Preserving Topology and Elasticity for Embedded Deformable Models. ACM Transaction on Graphics (proc. of SIGGRAPH 2009) .

Nesme, M., Payan, Y., Faure, F., 2006. Animating shapes at arbitrary resolution with non-uniform stiffness, in: 3rd Workshop in Virtual Reality Interaction and Physical Simulation, VRIPHYS '06, Madrid, Spain.

Nienhuys, H.W., van der Stappen, A.F., 2001. Supporting cuts and finite element deformation in interactive surgery simulation. Technical Report. University of Utrecht.

Sifakis, E., Der, K.G., Fedkiw, R., 2007. Arbitrary cutting of deformable tetrahedralized objects, in: SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland. pp. 73–80.

Wicke, M., Botsch, M., Gross, M., 2007. A Finite Element Method on Convex Polyhedra, in: Proceedings of Eurographics '07.