GPU-based Real-Time Soft Tissue Deformation with Cutting and Haptic Feedback

Hadrien Courtecuisse[a], Hoeryong Jung[a,b], Jérémie Allard[a], Christian Duriez[a], Doo Yong Lee[b], Stéphane Cotin[a]

[a]*SHAMAN Project, INRIA, France*
[b]*Robotics and Simulation Laboratory, KAIST, Korea*

Abstract

This article describes a series of contributions in the field of real-time simulation of soft tissue biomechanics. These contributions address various requirements for interactive simulation of complex surgical procedures. In particular, this article presents results in the areas of soft tissue deformation, contact modelling, simulation of cutting, and haptic rendering, which are all relevant to a variety of medical interventions. The contributions described in this article share a common underlying model of deformation and rely on GPU implementations to significantly improve computation times. This consistency in the modelling technique and computational approach ensures coherent results as well as efficient, robust and flexible solutions.

Keywords: biomechanics, soft tissue, real-time, finite element method, GPU, haptic rendering

1. Introduction

During the past decade, a number of new minimally invasive surgical techniques have been introduced in an effort to reduce patient's pain, recovery time, and in some cases, operating time. One of the most important changes has been laparoscopic surgery, which brought new technologies into the operating room and created a distance between the surgeon and the patient. More recently, other minimally invasive techniques have been proposed, such as natural orifice transluminal endoscopic surgery, which can be considered as an evolution of laparoscopic surgery. As a new surgical technique, laparoscopy requires surgeons to acquire new skills, and adapt to changes from conventional open surgery (e.g. amplified tremor, diminished tactile sensation, loss of depth perception). This has been a motivation for a number of works in the field of surgery simulation, real-time deformable models, or haptic rendering (see for instance Marescaux et al. (1998), Picinbono et al. (2000), Brown et al. (2002), Forest et al. (2004), or Harders (2008)). As a result, it has been demonstrated that the use of computer-based simulators can lead to a more effective and systematic training, thus providing objective assessment of technical competence (Seymour et al., 2002). Other studies also show that the skills learned thanks to a simulation can be transfered into the operating room (see Grantcharov et al. (2004) for instance).

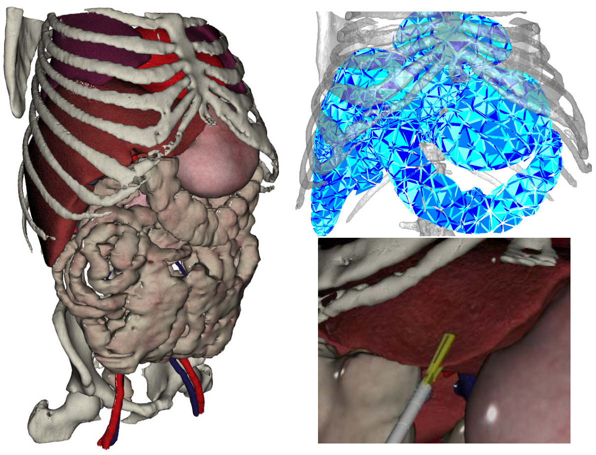


Figure 1: *GPU-based simulation of multiple anatomical structures reconstructed from a CT dataset. The simulation is based on a series of methods proposed in this article and involves implicit co-rotational FEM, frictional contacts, and cutting with haptic feedback.*

More recently, a new thrust has appeared with the development of medical imaging techniques which could make it possible to develop patient-specific simulations. Such simulations could be beneficial in certain situations, when the patient presents a rare pathology or when the best surgical strategy is unclear. In this case the simulation can be used as an efficient planning tool, however more realistic models of the patient's anatomy and behaviour are required. A final step concerns the use of augmented reality systems for image guided surgery, to improve the accuracy and limit the adverse effects of surgery. In order to register pre-operative data on the real organs (to visualize the targeted area while the procedure is progressing for instance) accurate, real-time biomechanical models are needed, but their interaction with medical devices also needs to be modelled. Such interactions not only involve tissue manipulation, but also tissue dissection.

In this context, the development of fast algorithms to compute the deformation, contact response, cutting and haptic feedback of soft tissues could enable a number of the aforementioned applications. More specifically, when considering requirements for realistic interactive simulations of medical procedures, several elements seem mandatory: anatomical models and tissue properties need be patient-specific and obtained without complex additional procedure; soft tissue behaviour needs to be realistic and demonstrate a predictive capability, yet it should be compatible with real-time computation; interactions with the surrounding anatomy and with medical devices need to involve advanced contact models that can be be computed in real-time; the different types of dissection performed on soft tissues should be simulated; and finally realistic visual and haptic feedback should be provided to create a higher level of immersion, in particular during training sessions.

1.1. Previous Work

Among the numerous publications in the field of biomechanics, real-time deformable models, collision detection, contact modelling or haptics, few methods have been proposed to address all (or at least a majority) of the requirements listed above. Among the existing approaches which at least partially aim at this objective, we can cite methods based on spring-mass networks, methods based on linear elasticity, and explicit finite element models for non-linear materials.

Mass-spring networks: one of the most popular methods for real-time computation of the deformation of soft objects is based on spring-mass networks (see Montgomery et al. (2002) for instance). In addition, such an approach is well suited to benefit from GPU-based acceleration, as demonstrated by (Sorensen et al., 2006). Another obvious advantage of using spring-mass networks is their ability to handle topological changes at a reduced computational cost. Yet, although they are quite simple to implement and very fast to compute, spring-mass methods fail to properly characterize soft tissues deformation as they introduce artificial anisotropy through the choice of the mesh, and make it difficult to relate spring stiffness to material properties such as Young modulus. A recent work by Delingette (2008) has shown that such a relationship can be established in the case of hyperelastic constitutive laws. In this case, it is possible to compute springs stiffness on triangular and tetrahedral meshes relating to St Venant-Kirchhoff materials.

Linear elasticity with a finite element method: there are several instances in the literature of real-time simulation which rely on linear elasticity. Initial work (see Bro-Nielsen and Cotin (1996), Cotin et al. (1999) or James and Pai (1999)) made the assumption of small displacements and often relied on pre-computed responses based on Finite Element Models (FEM) and related techniques. The main benefit of approaches based on pre-computation is the important speed up that can be obtained, allowing not only real-time deformations but also haptic feedback. However, two main issues arise from such approaches, the first one is the impact of cutting on the pre-computed response, the second one is the inherent limitation of the small strain assumption. Some solutions have been proposed to solve the first problem, using for instance numerical techniques to update a pre-inverted stiffness matrix (Lee et al., 2005). The small strain limitation is known to lead to incorrect results (in particular when rotations are applied). An elegant solution for this problem was introduced by Felippa (2000) and is known as the co-rotational method. Other methods have been proposed to extend the idea of pre-computation, but in the case of geometrically non-linear deformations (Barbič and James, 2005) (Mahvash and Hayward, 2004).

Explicit non-linear FEM: another strategy to deal with real-time computation of soft-tissue deformations when relying on a finite element approach is to base the computation on an explicit integration scheme, as proposed by Taylor et al. (2008) for instance. The main advantage is that the solving process only involves the mass matrix, which is diagonal if mass lumping is used. Thus, the equations of motion can be decoupled and each degree of freedom can be solved independently. The solving process is then very quick and its parallelization is quite straightforward (Comas et al., 2008). Explicit integration methods are particularly well suited for applications such as real-time non-rigid registration of brain shift during surgery (Joldes et al., 2009b). In this case, only the steady state of the deformation is sought and the mass can be artificially increased in order to deal with stiff materials (Joldes et al., 2009a).

However, our goal is different. We aim at simulating complex, user-controlled interactions between medical devices and anatomical structures that are often deformable. This leads to complex interactions which are unpredictable and discontinuous in time (non smooth contact problems for instance). Additionally, such simulations are dynamic by nature, and may involve haptic feedback and topological changes. Based on these requirements, the choice of an implicit integration scheme offers the best tradeoff between robustness, stability, convergence and computation time, in particular when combined with a GPU implementation. Although this choice leads to added difficulties compared to an explicit approach, we show in this paper that it can be at the center of a framework which addresses all the requirements of interactive simulations.

1.2. Many-cores Architectures

In recent years, the computational hardware available in high-performance workstations shifted from increasingly efficient but complex sequential computational units, to smaller units, each not much faster than previous generations, but duplicated to be able to execute more threads in parallel. This evolution has taken place both in the design of CPUs and recent Graphics Processing Units (GPUs). The latest generation of GPUs contains hundreds of computation units (240 in NVIDIA Geforce 285 GTX, 1600 in ATI Radeon 5870). This radical architectural change has important consequences on the type of algorithms which are applicable in interactive simulations.

In terms of programming, general purpose computations on GPUs initially required the use of graphics-oriented libraries. Recently, the two major GPU vendors released general programming APIs, CUDA (Nickolls et al., 2008) and CTM (Peercy et al., 2006) which provide direct access to the underlying parallel processors of the GPU, as well as full instruction sets, such as double precision computations and write operations at arbitrary locations. In 2009, a multi-vendor standard, named OpenCL (Munshi, 2008), was released, with a programming model very similar to CUDA. In the following sections we present a series of algorithms which have been implemented in CUDA. It is relatively straightforward to also implement them in OpenCL if necessary.

1.3. Summary of the Contributions

In this article we introduce a suite of methods which rely on a common underlying model to obtain a coherent answer to the various requirements of an interactive simulator. For this we choose a non-linear corotational model with implicit integration (section 2), associated with a methodology for handling contact and stable haptic rendering (section 3), as well as an efficient approach to simulate cutting (section 4). Finally we show GPU implementations of these methods which produce a fast and stable simulation of soft tissue deformation. These contributions are illustrated in a simulation of laparoscopic hepatectomy (section 5).

2. GPU Implementation of Implicit Deformations

The behaviour of deformable structures such as soft tissues is described by a constitutive law which needs to be solved using a numerical technique. As we also consider the dynamics of the bodies, an integration scheme also needs to be chosen. There is a relatively large gap between models typically used in the field of biomechanics and approaches traditionally chosen to obtain real-time computation of deformable bodies. An approach which is gaining a lot of ground in real-time soft tissue deformation is the combination of linear elastic material with a finite element method (FEM). Although more complex to implement, FEM methods are the usual choice in biomechanics for numerically solving the partial differential equations of constitutive laws. The simplest model is to use a linear stress-strain relationship. However, such models are not invariant by rotation, i.e. simply rotating an object will create non-null forces. The co-rotational method, introduced by Felippa (2000) provides a relatively simple mean to handle large displacements and geometrical non-linearities.

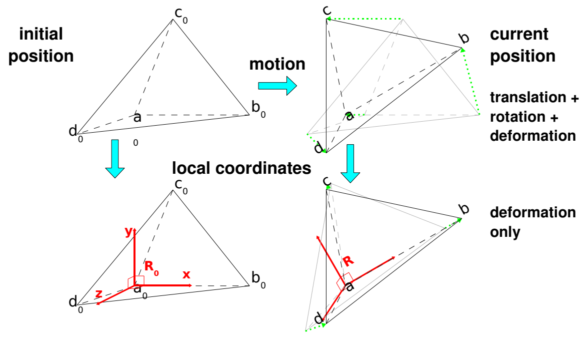


Figure 2: *Co-rotational FEM: a local frame is computed on each element to estimate the local rigid motion.*

2.1. Equations of Motion for Deformable Objects

An expression of the internal and external forces applied to the simulated objects depending on the current state is given as $\mathbf{f}(\mathbf{x}, \mathbf{v})$, where $\mathbf{x}, \mathbf{v}, \mathbf{f}$ are respectively the position, velocity and force vectors. The acceleration $\mathbf{a}$ can then be defined using the *mass* matrix $\mathbf{M}$ and Newton's second law: $\mathbf{M} \cdot \mathbf{a} = \mathbf{f}(\mathbf{x}, \mathbf{v})$. This introduces a non-linear ordinary differential equation system.

Implicit schemes provide several advantages, in particular improved stability with relatively large time steps. This is particularly relevant for interactive simulations involving contacts with virtual devices controlled by an operator. Using implicit time integration schemes requires however to compute the solution of a non-linear system of equations at each time step. For instance, a backward Euler scheme, which updates velocities and positions based on accelerations at the end of the time step, can be described as:

$$\mathbf{v}_{t+h} = \mathbf{v}_t + h\, \mathbf{a}_{t+h} \qquad \mathbf{x}_{t+h} = \mathbf{x}_t + h\, \mathbf{v}_{t+h} \tag{1}$$

$$\mathbf{M} \cdot \mathbf{a}_{t+h} = \mathbf{f}(\mathbf{x}_{t+h}, \mathbf{v}_{t+h}) \tag{2}$$

As the forces at the end of the time step are not known, a first-order approximation is used:

$$\mathbf{f}(\mathbf{x}_{t+h}, \mathbf{v}_{t+h}) \approx \mathbf{f}(\mathbf{x}_t, \mathbf{v}_t) + \mathbf{K} \cdot (\mathbf{x}_{t+h} - \mathbf{x}_t) + \mathbf{B} \cdot (\mathbf{v}_{t+h} - \mathbf{v}_t) \tag{3}$$

where $\mathbf{K}$ is the *stiffness* matrix and $\mathbf{B}$ the *damping* matrix. Substituting (1) and (3) into (2) provides the final linearized system :

$$\underbrace{\left(\mathbf{M} - h\mathbf{B} - h^2\mathbf{K}\right)}_{\mathbf{A}} \cdot \mathbf{dv} = \underbrace{h\, \mathbf{f}(\mathbf{x}_t, \mathbf{v}_t) + h^2\, \mathbf{K} \cdot \mathbf{v}_t}_{\mathbf{b}} \tag{4}$$

where $\mathbf{dv} = h\, \mathbf{a}_{t+h} = \mathbf{v}_{t+h} - \mathbf{v}_t$.

In our approach, the system of equations defined by the co-rotational FEM is solved iteratively using a Filtered Conjugate Gradient method (Baraff and Witkin, 1998). This iterative method can be tuned to trade-off accuracy for speed by controlling the number of iteration and residual error threshold. In practice, only a few tens of iterations are required to obtain a stable and visually satisfying simulation. Interestingly, the matrix needs not be explicitly computed, as only matrix-vector multiplications and vector dot products are applied in the iterative solution.

2.2. Parallelization on GPU

This section describes the main parallelization and optimization techniques used to implement on GPU the mechanical computations as used in our framework.

Forces Computation: to parallelize a given set of computations on a GPU it is necessary to extract a massive level of parallelism, on the order of tens of thousands of tasks.

Our approach is to avoid of computing a large sparse system matrix such as in equation (3). Instead, we compute for each element e its local stiffness matrix $\mathbf{K_e}$, and we obtain the local forces with :

$$\mathbf{f_e} = \mathbf{R_e} \cdot \mathbf{K_e} \cdot \mathbf{R_e^T} \cdot \mathbf{u_e} \tag{5}$$

Where $\mathbf{u_e}$ the displacement vector and $\mathbf{R_e}$ the local rotation of an element. All these computations are independent and are straightforward to parallelize. However we then need to accumulate these internal forces from FEM elements to vertices which are shared by multiple elements. Thus, computation threads need to scatter their results on a set of shared variables. Such operations can be represented as a graph, where nodes represent shared variables and edges computations between them.

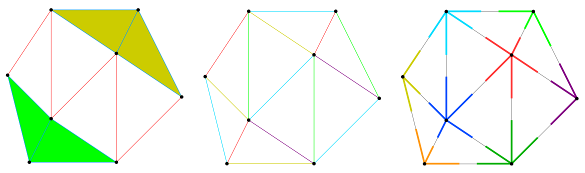
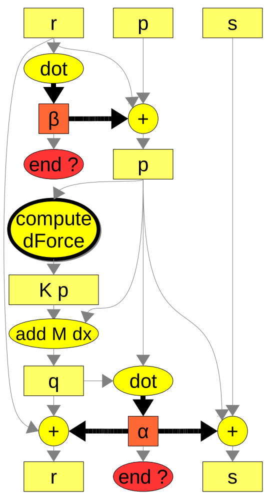


Figure 3: *Parallel write conflicts can be removed by (a) mesh partitioning, (b) graph coloring or (c) transforming scatter into gather.*

A very common technique to parallelize such graphs is to partition it into a set of subgraphs (Fig. 3a), each computed by a different processor. The computations on the border between partitions will have to be handled specifically, creating overheads. This method is efficient only if the number of processors is small compared to the number of nodes in the graph. An alternative method, *graph-coloring* (Fig. 3b), partition the graph into sub-graphs such as a node is never shared between edges. This allow to execute all edges within each subgraph in parallel, but requires $n - 1$ synchronizations if n partitions (or colors) are necessary.

Finally, another method consists in transforming the parallel scatter operation into a parallel gather. Instead of computing edges in parallel, the nodes are processed in parallel, each gathering the results from the connected edges (Fig. 3c). This requires either duplicating the computations of each edge in the threads of the two connected nodes, or using a temporary buffer to store the result of each edge computation, and then doing the gather step reading from this buffer. This final method is used to implement the FEM forces computation :

1. First, using one thread per tetrahedron, all per-element computations are done in parallel, and the



```
$r = b$ // initial residual with
$s = 0$
$\delta_0 = dot(r, r)$
$i = 1$
while $i <= i_{max}$ do
    $\delta_i = dot(r, r)$
    if $\delta_i \leq \epsilon^2 \delta_0$ then $break$
    $\beta = \frac{\delta_i}{\delta_{i-1}}$
    $p = \beta\, p + r$
    $q = M * p - h^2 K * p$
    $den = dot(p, q)$
    if $den \leq threshold$
    then $break$
    $\alpha = \frac{\delta_i}{den}$
    $s = s + \alpha\, p$
    $r = r - \alpha\, q$
    $i = i + 1$
```

Figure 4: *Task graph and algorithm for GPU Conjugate Gradient. Squares represent data vectors, while circles represent computation tasks. GPU tasks are shown in yellow whereas CPU tasks are shown in red. Arrows define data dependencies, from which we can deduce the required ordering for the computations and data transfers. To highlight important steps affecting execution speed, data transfers between the CPU and GPU as well as computationally intensive tasks are shown in* ***bold****.*

result is stored independently for each element in the temporary vector.

2. Then, using one thread per vertex, the forces on each vertex are accumulated, using a pre-computed array storing for each vertex the indices of all connected tetrahedron.

Parallel Implicit Solver: implicit integration schemes require a complex architecture, as one or more linear systems need be solved. Other than simple linear vector algebra, only matrix–vector products are required, which can be implemented directly without actually computing or storing the matrix itself. This is illustrated in Figure 4, describing a backward Euler implicit integration using the Conjugate Gradient algorithm. *Compute dForce* and *add M dx* tasks correspond to the multiplication of a vector by the respective stiffness and mass matrices. Two scalar values, resulting from dot product operations, must be transmitted back to the CPU at each iteration to evaluate the convergence criteria. This limited data exchange between GPU and CPU helps maintain a very efficient computational performance.

2.3. Performance Evaluation

When using the proposed approach for computing the dynamics of a deformable object, we obtain speed-ups between 15× and 35× on a GeForce GTX 280 compared to a sequential implementation on a Core 2 Quad at 3.00 GHz (see Figure 5). This test was performed on a mesh with $41,000$ nodes, with simulation times of 1.4 FPS on the CPU and 46.15 FPS on GPU.

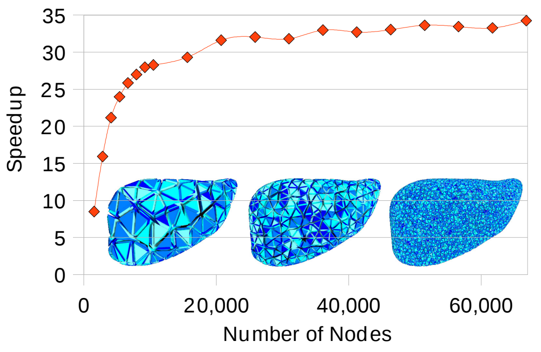


Figure 5: *Computational speed-up obtained with the GPU version of a co-rotational implicit FEM models.*

3. Modelling Complex Interactions

Whether they are targeted for training, planning or per-operative guidance, the simulations we consider need to account for interactions with the surrounding anatomy or with medical devices. While the biomechanical behaviour of soft-tissues has been well studied, a few works exist regarding the mechanical interactions between anatomical structures. These interactions are not always of the same nature: interaction types include contacts with friction, direct mechanical connections, or links through connective tissues, for instance. For tissue-tool interactions, most approaches rely on a simple contact models, and rarely account for friction. The way contacts are handled remains an open problem (Acary and Brogliato, 2008) even if a significant amount of work can be found on this subject (Johnson, 1985; Wriggers, 2002). All these interactions play a key role in the overall behaviour of soft-tissues and therefore the way they are processed highly influences the post-impact motion of the interacting objects.

This section offers an overview of our approach and shows how to optimize the computation for co-rotational models. A GPU implementation is also described, in order to provide another means to improve computational efficiency. Finally, we also show how optimal haptic rendering can be obtained using the proposed method.

3.1. Constraint-based Approach

In this work, the various interactions are modelled using a set of constraints. For each constraint, we assign a law, which depends on the relative position of interacting objects[1]:

$$\Phi(\mathbf{x}_1, \mathbf{x}_2) = 0 \tag{6}$$

$$\Psi(\mathbf{x}_1, \mathbf{x}_2) \geq 0 \tag{7}$$

Where Φ represents the bilateral interaction laws (attachement, sliding...) between object whereas Ψ represents unilateral interaction laws (contact, friction...). These functions can be non-linear.

To solve these laws, we use a Lagrange Multipliers approach and a single linearization by time step. For both interacting objects, equation 4 is then replaced by:

$$\begin{aligned} \mathbf{A}_1\mathbf{dv}_1 &= \mathbf{b}_1 + \mathbf{H}_1^T\boldsymbol{\lambda} \\ \mathbf{A}_2\mathbf{dv}_2 &= \mathbf{b}_2 + \mathbf{H}_2^T\boldsymbol{\lambda} \end{aligned} \tag{8}$$

where $\mathbf{H}_1 = [\frac{\delta\Phi}{\delta\mathbf{x}_1}\ ;\ \frac{\delta\Psi}{\delta\mathbf{x}_1}]$ and $\mathbf{H}_2 = [\frac{\delta\Phi}{\delta\mathbf{x}_2}\ ;\ \frac{\delta\Psi}{\delta\mathbf{x}_2}]$ and $\boldsymbol{\lambda}$ is the unknown force vector applied to solve the constraints. In order to solve $\boldsymbol{\lambda}$ the process is performed by the following steps.

Step 1 : interacting objects are solved independently while setting $\boldsymbol{\lambda} = 0$. We obtain what we call the *free motion* $\mathbf{dv}_1^{\text{free}}$ and $\mathbf{dv}_2^{\text{free}}$ for each object. After integration, we obtain $\mathbf{x}_1^{\text{free}}$ and $\mathbf{x}_2^{\text{free}}$.

Step 2 : the constraint laws are linearized:

$$\underbrace{\begin{bmatrix}\Phi(\mathbf{x}_{1,t+h}, \mathbf{x}_{2,t+h}) \\ \Psi(\mathbf{x}_{1,t+h}, \mathbf{x}_{2,t+h})\end{bmatrix}}_{\delta_{t+h}} = \underbrace{\begin{bmatrix}\Phi(\mathbf{x}_1^{\text{free}}, \mathbf{x}_2^{\text{free}}) \\ \Psi(\mathbf{x}_1^{\text{free}}, \mathbf{x}_2^{\text{free}})\end{bmatrix}}_{\delta^{\text{free}}} + h\mathbf{H}_1\mathbf{dv}_1^{\text{cor}} + h\mathbf{H}_2\mathbf{dv}_2^{\text{cor}} \tag{9}$$

With $\mathbf{dv}_1^{\text{cor}}$ and $\mathbf{dv}_2^{\text{cor}}$ being the unknown corrective motion when solving equation 8 with $\mathbf{b}_1 = \mathbf{b}_2 = 0$. When gathering equations 8 and 9, we have:

$$\delta_{t+h} = \delta^{\text{free}} + h\left[\mathbf{H}_1\mathbf{A}_1^{-1}\mathbf{H}_1^T + \mathbf{H}_2\mathbf{A}_2^{-1}\mathbf{H}_2^T\right] \tag{10}$$

We obtain the value of $\boldsymbol{\lambda}$ using a projected Gauss-Seidel algorithm which iteratively verifies and projects the various constraint laws contained in Φ and Ψ (see Duriez et al. (2006) for details).

Step 3 : When the value of $\boldsymbol{\lambda}$ is available, the corrective motion is computed:

$$\begin{aligned} \mathbf{x}_{1,t+h} &= \mathbf{x}_1^{\text{free}} + h\mathbf{dv}_1^{\text{cor}} \ \text{ with }\ \mathbf{dv}_1^{\text{cor}} = \mathbf{A}_1^{-1}\mathbf{H}_1^T\boldsymbol{\lambda} \\ \mathbf{x}_{2,t+h} &= \mathbf{x}_2^{\text{free}} + h\mathbf{dv}_2^{\text{cor}} \ \text{ with }\ \mathbf{dv}_2^{\text{cor}} = \mathbf{A}_2^{-1}\mathbf{H}_2^T\boldsymbol{\lambda} \end{aligned} \tag{11}$$

[1]For simplicity, we present the equations for two interacting objects 1 and 2, but the method applies to any number of interacting bodies.

Equation 11 involves the inverse of matrix $\mathbf{A}$, which changes at every time step. Computing this inverse is obviously too time consuming for real-time simulation as soon as the objects involve more than a few hundred nodes. Moreover, when dealing with multiple contact points, we need to detect all colliding points and solve the response while taking into account contact and friction laws. The following sections address these two problems by taking advantage of some properties of corotational models.

3.2. Compliance Warping

The computation of the matrix in equation 10 requires the computation of matrix $h\mathbf{A}^{-1}$ of the interacting objects. This matrix is homogeneous to a compliance, i.e. the inverse of a stiffness. With deformable objects undergoing large displacements, $\mathbf{A}^{-1}$ must be updated when $\mathbf{M}$, $\mathbf{K}$ and $\mathbf{B}$ change according to the non-linear deformations. This computation is highly time-consuming.

However, using the FEM based corotational model presented above, we can obtain a good approximation of the matrix $\mathbf{A}^{-1}$ using its value in the rest shape configuration $\mathbf{A}_0^{-1}$. First, at the beginning of the simulation, we pre-compute the matrix $\mathbf{A}_0^{-1}$. Then, as the model account for the non-linear rotation which is induced by the deformation, we evaluate a rotation at each node, from its rest shape to its current shape. Finally, we approximate the value of the compliance matrix $\mathbf{C} \approx h\mathbf{A}^{-1}$ using the following equation:

$$\mathbf{C} = h\mathbf{R}\mathbf{A}_0^{-1}\mathbf{R}^T \tag{12}$$

Where $\mathbf{R}$ is a 3×3 block diagonal matrix which gathers the rotations associated with object nodes. The rotation evaluation is done using an average of the frames computed for the elements in the direct neighborhood of the point (see section 2.1). This simplification significantly speeds up the computation of the matrix of Equation 10. The speed up is enforced by using the sparsity of matrices $\mathbf{R}$ (diagonal matrix) and $\mathbf{H}$.

The same approximation is used when applying corrective motion. Equation 11 becomes (for object 1, for instance):

$$\mathbf{x}_{1,t+h} = \mathbf{x}_1^{\text{free}} + \mathbf{C}\ \mathbf{H}_1^T\boldsymbol{\lambda} \tag{13}$$

3.3. Contact Force Computation

The computation of multiple contact forces is a central application of compliance warping method. In respect of the frictionless contact case, we build and solve the following Linear Complementarity Problem (LCP):

$$\begin{cases} \delta_{t+h} = \underbrace{\mathbf{H}_1\mathbf{C}_1\mathbf{H}_1^T + \mathbf{H}_2\mathbf{C}_2\mathbf{H}_2^T}_{\mathbf{W}} \lambda + \delta^{\text{free}} \\ 0 \leq \delta_{t+h} \perp \lambda \geq 0 \end{cases} \tag{14}$$

Where **W** is the warped matrix of the contacts compliance. Several fast solvers can provide a solution for this LCP (Murty, 1997). The solution guaranties Signorini's law: no interpenetration and well oriented forces. We add friction using the Coulomb's model: each contact reaction force lies within a spacial conical region whose height and direction is given by the normal force. Several papers describe how to extend a LCP in order to include and solve friction (see, for instance Anitescu et al. (1999) and Jourdan et al. (1998)). Our implementation relies on the iterative Gauss-Seidel approach described in (Duriez et al., 2006).

3.4. Corrective Motion

When the iterative solver has converged, every constraint law condition is fulfilled: contact and friction laws are followed at each contact point and the obtained motions avoid disturbing interpenetration. However, as we use **W**, we rely on an approximate compliance. It means that the motion created by these constraints is not totally accurate even if the compliance warping provides a very good approximation.

To get a post-contact behavior that is coherent with the LCP, we compute the constrained motion using the same compliance warping method. The steps involved in the corrective motion are:

1. We map the contact forces in the rest shape coordinate frame: $\mathbf{r}_0 = (\mathbf{HR})^T\lambda$
2. We compute the displacement in the original coordinate: $d\mathbf{x}_0 = h\mathbf{A}_0^{-1}\mathbf{r}_0$
3. We rotate back the displacement to the current coordinate frame: $d\mathbf{x} = \mathbf{R}d\mathbf{x}_0$
4. We find the new position: $\mathbf{x}_{t+h} = \mathbf{x}^{\text{free}} + d\mathbf{x}$

These steps are processed for both interacting object 1 and 2 and can be executed in parallel.

3.5. GPU Implementation

The constraint process requires the computation of **W** at each time step of the simulation. The size of this square matrix depends on the number of constraints simultaneously involved. Its computation could quickly become a bottleneck of real-time execution when a lot of contacts are involved. Fortunately, the majority of required operations can be massively parallelized.

One issue is to avoid writing conflicts: interacting objects need to add data at the same location in the compliance matrix. To solve this problem, each object computes its own local compliance matrix $\mathbf{W}_L$ independently. Then we sum these sequentially to build **W**. Replacing (12) into (10) provides the final system :

$$\mathbf{W} = \sum \underbrace{h(\mathbf{HR})\mathbf{A_0}^{-1}(\mathbf{HR})^T}_{W_L} \tag{15}$$

Therefore, using 2 kernels per object we can build global compliance matrix as follow :

1. $\mathbf{H_r} = \mathbf{HR}$
2. $\mathbf{W} = \mathbf{W} + h\mathbf{H_r}\mathbf{A_0}^{-1}\mathbf{H_r^T}$

Another issue is to select an efficient storage method for dense and sparse matrices, as the performance of GPU kernels is highly impacted by non regular memory access. Indeed, **H** is a 3×3 block sparse matrix, **R** is a 3×3 block-diagonal matrix and $\mathbf{A_0}^{-1}$ is a dense matrix. We use a vector to store **R** and a full matrix for $\mathbf{A_0}^{-1}$. Matrix **H** is highly sparse with non-zero values only at the degrees of freedom influenced by contacts. To improve the compatibility of this highly sparse matrix with GPU computing, we use a Jagged Diagonal Storage (Barrett et al., 1994). This allows all threads in the same group of the first kernel to access aligned-data on the same line, and just a local gathering operation is necessary.

For the second kernel, we assign to each thread the computation of one element of **W**, and we create groups of threads for each line of the matrix. Thereby, in the same group, the threads read to the same data in the matrix **H**. We use GPU shared memory to accelerate the computation. Finally, we add all values in the global compliance matrix by reading the rows and columns addresses of **H**. Once the matrix is built, we solve equation (14) using Gauss-Seidel algorithm. We can choose either to solve it while bringing back data on classical CPU or to solve it directly on GPU (Courtecuisse and Allard, 2009). We find that the Gauss-Seidel is more efficient on GPU if the number of constraints is high (more than a thousand).

3.6. Haptic Rendering

Precise modelling of mechanical interactions with virtual instruments is essential to obtain both a realistic behaviour and convincing haptic feedback. Haptic rendering requires a high rate (from 300Hz to 1000Hz) for force computation. For this we rely on the multi-rate haptic rendering technique presented in Saupin et al. (2008), which separates the haptic loop from the simulation loop.

When the instruments create deformations on soft tissues they modify the boundary conditions of the models and consequently modify their behaviour. We propose a physically-based model of interaction which accounts for the mass $\mathbf{M}_{\text{hap}}$ of the instrument as well as its stiffness $\mathbf{K}_{\text{hap}}$ and damping $\mathbf{B}_{\text{hap}}$ (corresponding to the gain in position and velocity of the control loop of the haptic device). When the instrument is deformable (for instance a flexible endoscope), matrix $\mathbf{K}_{\text{hap}}$ also includes the deformation model of the instrument. Based on $\mathbf{M}_{\text{hap}}$, $\mathbf{B}_{\text{hap}}$ and $\mathbf{K}_{\text{hap}}$, we compute a compliance matrix $h\mathbf{A}_{\text{hap}}^{-1}$ for the virtual instrument interacting with the soft-tissues. This way, the proposed haptic rendering model follows the same constraint-based modelling introduced above. At each time step of the simulation loop, we build the LCP using matrix $\mathbf{W}$, and compute the contact force $\boldsymbol{\lambda}$ from the LCP. The LCP and contact force are then transmitted to the haptic loop to compute inter-sampled forces during each time step of the simulation loop. In the haptic loop, the value of $\boldsymbol{\delta}^{\text{free}}$ is updated using new position $\mathbf{x}_{\text{hap}}$ of haptic device. A new computation of the LCP is then performed which provides inter-sampled contact forces $\boldsymbol{\lambda}$ that can be displayed on the haptic device for a stable and very fast rendering.

3.7. Performance Evaluation

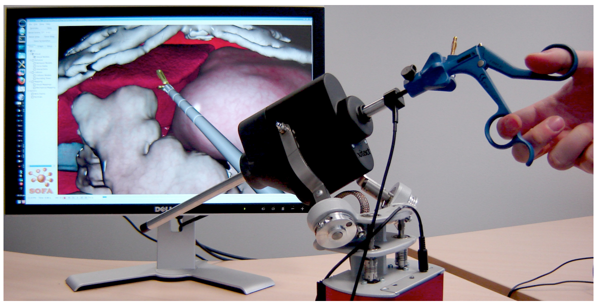


Figure 6: *Simulated grasping of the liver using a laparoscopic haptic device. During this interaction, the liver can be in contact with the ribs, colon or stomach.*

To illustrate the benefit of the compliance warping approach in the context of the real-time simulation and haptic rendering, we rely on a simulation of grasping with a liver model composed of about $1,300$ tetrahedra. The liver model as well as the surrounding anatomy are obtained from a generic anatomical dataset (see Figure 6). The virtual laparoscopic grasper is controlled by a Xitact IHP haptic device. During the interactions, the instrument is in contact with the liver and the liver can be in contact with other anatomical structures around it. Up to 150 contact points can be detected leading to up to 450 constraints. The simulation framerate is of 70 frames/sec on a CPU Intel i7 975 quad-core 3.33GHz while the haptic thread runs at a frequency of 1 kHz, leading to realistic and stable interactions.

If the same simulation is performed using a direct inversion[2] of the system matrix to compute the compliance, the computation time is increased by a factor 50×, which would limit the number of possible contact points to a maximum of 10 in order to maintain real-time computation.

4. Cutting

In this section, we extend our simulation methodology to real-time virtual cutting of soft-tissues. Simulating cutting requires to modify the mesh of the object and in response to update the system and compliance matrices used in our computations. To simplify the problem, we break down the topology changes arising from cutting into two simpler topology changes; element removal and element addition, and update the topology as a combination of these simpler changes. We compute the impact of each topological change on the corotational and warped compliance computations, as presented in the following sections.

4.1. Topology Modification

For FEM computation, the deformable model is meshed into a set of volume elements which are connected together by a topological map. The matrices $\mathbf{M}$, $\mathbf{B}$ and $\mathbf{K}$ presented above are built using this mesh structure. Generally, cutting induces some changes to the topological map as it disconnects a part of the object. To simulate a cut based on physics, the mesh and the matrices $\mathbf{M}$, $\mathbf{B}$ and $\mathbf{K}$ should be adequately modified to be consistent with the modified topology. Figure 7 shows the process we employ for the topology modifications. Rather than reconstructing the mesh and the matrices, we incrementally update them from the current state with three steps; element removal, element subdivision, and element addition. A specific set of edges of the virtual instrument is used to define cut-lines. Between times t_i and t_{i+1}, cut-lines define a cut-surface that can potentially intersect some elements of the deformable mesh.

[2]To do the comparison, we use a direct sparse LDL solver that is highly optimized for this type of computation.

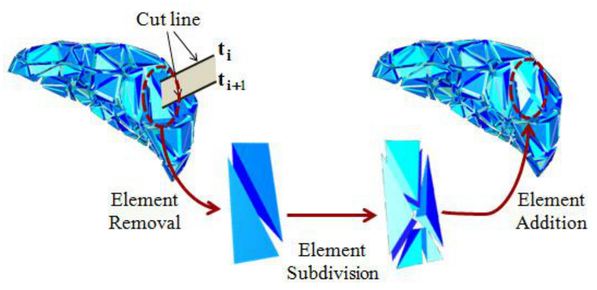


Figure 7: *The soft-tissue mesh is cut and the topology is incrementally updated.*

Element removal: first, we remove the intersected elements from the overall mesh of the object and then we subtract the contribution of the removed elements from the current matrices. For instance, the stiffness matrix $\mathbf{K}$ is updated to $\mathbf{K}_u = \mathbf{K}_c - \sum_{e=1}^{N_r} \mathbf{G}_e \mathbf{K}_e \mathbf{G}_e^T$, where $\mathbf{K}_c, \mathbf{K}_u$ represent respectively the current and updated stiffness matrix, N_r is the number of removed elements, $\mathbf{K}_e$ is the element stiffness matrix, and $\mathbf{G}_e, \mathbf{G}_e^T$ are the globalization matrices which map the rows and the columns of the element matrix to the global matrix. The mass and damping matrices are also updated in a similar way.

Element subdivision: secondly, we subdivide the removed elements so that they can be aligned with the cut-surface. This step is performed using the element subdivision algorithm proposed by (Mor and Kanade, 2000). In this step, several additional nodes are introduced to build the new volumetric elements. The dimension of the matrices $\mathbf{M}$, $\mathbf{B}$ and $\mathbf{K}$ is extended to be consistent with the new number of nodes.

Element addition: lastly, we add the subdivided elements to the overall mesh of the object, and also add the contribution of added elements to the current matrices $\mathbf{M}$, $\mathbf{B}$ and $\mathbf{K}$. In this step, the contribution of the new elements is added to the matrices as explained in the element removal.

4.2. Compliance Matrix Update

As stated in section 3, the contact model relies on an update of the compliance matrix computed for the initial configuration of the mesh. As virtual cutting takes place, this matrix is no longer valid and the pre-computed compliance matrix should be recomputed. However, a direct computation would involve the inversion of large matrix $\mathbf{A}_0 = (\mathbf{M}_0 + h\mathbf{B}_0 + h^2\mathbf{K}_0)$, which is computationally too demanding to be performed in real-time. Since topological changes caused by the cutting only alters a few entries of the matrices $\mathbf{M}_0$, $\mathbf{B}_0$ and $\mathbf{K}_0$, we propose to use a low rank inverse update algorithm, as a reasonable solution for real-time computation.

4.2.1. Low Rank Modification

Once the topology of the deformable object is modified by cutting, we update the matrix[3] $\mathbf{A}^i = (\mathbf{M}_0^i + h\mathbf{B}_0^i + h^2\mathbf{K}_0^i)$ to obtain a new $\mathbf{A}^{i+1}$ where the superscript i represents the $i'th$ modification. We define two types of modification of the matrix $\mathbf{A}^i$: *low rank correction* involves a modification during which the size of the matrix remains constant and *low rank extension* involves an increase of the matrix size. Consequently, the updated matrix $\mathbf{A}^{i+1}$ can be divided into 4 block matrices:

$$\mathbf{A}^{i+1} = \begin{bmatrix} \mathbf{A}_{11}^{i+1} & \mathbf{A}_{12}^{i+1} \\ \mathbf{A}_{21}^{i+1} & \mathbf{A}_{22}^{i+1} \end{bmatrix} \tag{16}$$

The block matrix $\mathbf{A}_{11}^{i+1}$, the dimension of which is exactly the same as matrix $\mathbf{A}^i$, corresponds to the low rank correction, and is computed by adding and subtracting the contribution of the added and removed elements to the current matrix $\mathbf{A}^i$:

$$\mathbf{A}_{11}^{i+1} = \mathbf{A}^i + \mathbf{G}\mathbf{S}\mathbf{G}^T \tag{17}$$

where $\mathbf{G}$, $\mathbf{G}^T$ are the globalization matrices and $\mathbf{S}$ is the correction matrix which is constructed by summing the contribution of removed and added elements. The dimension of the correction matrix $\mathbf{S}$ is proportional to the number of distinct nodes impacted by the removal and addition of elements and, usually, it is very small compared with that of the matrix $\mathbf{A}^i$. In particular, as the cutting process takes place over a period of time, the number of modified elements per time step can be bounded (see section 4.3.1). The block matrices $\mathbf{A}_{12}^{i+1}$, $\mathbf{A}_{21}^{i+1}$, $\mathbf{A}_{22}^{i+1}$ correspond to the low rank extension which is induced by the newly created nodes. The dimension of square matrix $\mathbf{A}_{22}^{i+1}$, is proportional to the number of added nodes during cutting, and it is also very small compared with the dimension of the matrix $\mathbf{A}^i$. To compute the updated inverse matrix $\mathbf{C}^{i+1} = (\mathbf{A}^{i+1})^{-1}$ according to the modified matrix $\mathbf{A}^{i+1}$, we employ two kinds of low rank matrix inverse update algorithms; Sherman-Morrison-Woodbury formula which corresponds to the low rank correction and Block-wise matrix inversion which corresponds to the low rank extension.

[3]In the following, matrix $\mathbf{A}$ is computed in the rest shape configuration: $\mathbf{A} = \mathbf{A}_0$, we remove the subscript for sake of clarity.

4.2.2. Low Rank Inverse Update

The updated compliance matrix can be expressed with 4 block matrices like the updated system matrix.

$$\mathbf{C}^{i+1} = \begin{bmatrix} \mathbf{C}_{11}^{i+1} & \mathbf{C}_{12}^{i+1} \\ \mathbf{C}_{21}^{i+1} & \mathbf{C}_{22}^{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11}^{i+1} & \mathbf{A}_{12}^{i+1} \\ \mathbf{A}_{21}^{i+1} & \mathbf{A}_{22}^{i+1} \end{bmatrix}^{-1} \tag{18}$$

Each block matrix can be computed using a block-wise matrix inversion approach, as follows:

$$\begin{aligned} \mathbf{C}_{11}^{i+1} &= (\mathbf{A}_{11}^{i+1})^{-1} + (\mathbf{A}_{11}^{i+1})^{-1}\mathbf{A}_{12}^{i+1}\mathbf{Q}^{-1}\mathbf{A}_{21}^{i+1}(\mathbf{A}_{11}^{i+1})^{-1} \\ \mathbf{C}_{12}^{i+1} &= (\mathbf{C}_{21}^{i+1})^{T} = -(\mathbf{A}_{11}^{i+1})^{-1}\mathbf{A}_{12}^{i+1}\mathbf{Q}^{-1} \\ \mathbf{C}_{22}^{i+1} &= \mathbf{Q}^{-1} \end{aligned} \tag{19}$$

where $\mathbf{Q} = (\mathbf{A}_{22}^{i+1} - \mathbf{A}_{21}^{i+1}(\mathbf{A}_{11}^{i+1})^{-1}\mathbf{A}_{12}^{i+1})$ is called as Schür complement of $\mathbf{A}_{11}^{i+1}$. To compute each block of the compliance matrix $\mathbf{C}^{i+1}$, first, we compute the inverse of the matrix $\mathbf{A}_{11}^{i+1}$ using Sherman-Morrison-Woodbury formula.

$$\begin{aligned} (\mathbf{A}_{11}^{i+1})^{-1} &= (\mathbf{A}^{i} + \mathbf{G}\mathbf{S}\mathbf{G}^{T})^{-1} \\ &= \mathbf{C}^{i} - \mathbf{C}^{i}\mathbf{G}(\mathbf{S}^{-1} - \mathbf{G}\mathbf{C}^{i}\mathbf{G}^{T})^{-1}\mathbf{G}^{T}\mathbf{C}^{i} \end{aligned} \tag{20}$$

Where $\mathbf{C}^{i} = (\mathbf{A}^{i})^{-1}$. Then, we compute the matrix $\mathbf{Q}$ and $\mathbf{Q}^{-1}$ using the matrix $(\mathbf{A}_{11}^{i+1})^{-1}$. Lastly, we compute all the blocks of $\mathbf{C}^{i+1}$ by multiplying and adding given matrices.

Even though the proposed low rank inverse update algorithm is computationally efficient, we propose a GPU implementation to achieve real-time computation on large meshes.

4.3. GPU Implementation

The previously described algorithm computes $(\mathbf{A}^{i+1})^{-1}$ using dense linear algebra operations, which can be very efficiently implemented on parallel architectures.

Sherman-Morrison-Woodbury update: this step is performed by first evaluating the correction matrix $\mathbf{S}$ on the CPU. The size of this matrix is small (it depends on the number of nodes influenced by topological changes) and can be transferred at a very limited cost to the GPU. Three GPU kernels are then used to update $(\mathbf{A}_{11}^{i+1})^{-1}$ as follows:

1. $\mathbf{U} = \mathbf{C}^{i}\mathbf{G}$
2. $\mathbf{T} = \mathbf{U}\mathbf{Q}_{m}^{-1}$ with $\mathbf{Q}_{m} = \mathbf{S}^{-1} - \mathbf{G}\mathbf{C}^{i}\mathbf{G}^{T}$
3. $(\mathbf{A}_{11}^{i+1})^{-1} = \mathbf{C}^{i} - \mathbf{T}\mathbf{U}^{T}$

Although the first two steps could be computed on the CPU, a GPU version has the benefit of avoiding additional data transfers between CPU and GPU. The third kernel corresponds to the most expensive computation and as such, needs to be optimized as much as possible.

Low-rank matrix extension: when elements are added, we start by building $\mathbf{A}_{22}^{i+1}$ on the CPU. Next, we use a kernel, as previously, to compute the result of $\mathbf{A}_{21}^{i+1}(\mathbf{A}_{11}^{i+1})^{-1}\mathbf{A}_{12}^{i+1}$ in order not to transfer $(\mathbf{A}_{11}^{i+1})^{-1}$. We store the result into a smaller matrix which is actually transferred to compute the Schür complement on CPU. We finally update the compliance matrix using three GPU functions. For this, we need a temporary matrix with the same size as $(\mathbf{A}_{11}^{i+1})^{-1}$ to resize the $\mathbf{C}^{i}$ matrix.

1. $\mathbf{B}_{12} = (\mathbf{A}_{11}^{i+1})^{-1}\mathbf{A}_{12}^{i+1}\mathbf{Q}^{-1}$
2. $\mathbf{B}_{11} = (\mathbf{A}_{11}^{i+1})^{-1} + \mathbf{B}_{12}\mathbf{A}_{21}^{i+1}(\mathbf{A}_{11}^{i+1})^{-1}$
3. $\begin{cases} \mathbf{C}_{11}^{i+1} = \mathbf{B}_{11} \\ \mathbf{C}_{12}^{i+1} = (\mathbf{C}_{21}^{i+1})^{T} = -\mathbf{B}_{12} \\ \mathbf{C}_{22}^{i+1} = \mathbf{Q}^{-1} \end{cases}$

4.3.1. Performance Evaluation

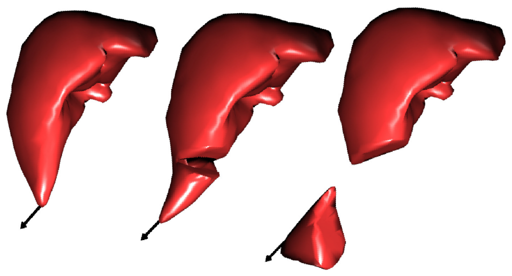


Figure 8: *Cutting simulation during interactive manipulation of a liver model composed of 3,874 tetrahedra.*

Two different liver meshes were used to evaluate our method, one composed of 1,607 tetrahedra and the second of 3,874. To ensure consistent measurements, a pre-defined cut-line was used to progressively remove a section of the organ. The compliance matrix was updated at each time step and computation times were recorded for each mesh. Computation times required to update the compliance matrix were evaluated on an Intel i7 975 quad-core CPU at 3.33 GHz and a Nvidia GPU GeForce GTX 285. Computations are performed using double precision on both architectures, as numerical instabilities could arise during matrix inversion operations when only simple precision is used.

Using the CUBLAS library to compute the most expensive kernels mentioned above, we manage to obtain a speedup close to 5×. Although not as important as in previous sections, this speed-up allows to use larger, more detailed, meshes during simulations. It should be noted that computation times depend on both the number of nodes in the mesh, and the number of nodes added or removed during each time step of the cutting operation. For instance, with a mesh composed of $1,607$ tetrahedral elements, only 22 msec are needed to update the compliance matrix if an average of 2 nodes are added or removed per time step. Similarly, 64 msec are needed for a mesh with $3,874$ nodes. Although computation cost highly grows with the number of elements cut in the same time step, in practice laparoscopic cutting is performed very progressively, therefore involving only a few elements per time step.

5. Simulation of Laparoscopic Hepatectomy

Every previously described algorithm has been implemented in the SOFA framework (Allard et al., 2007) in order to simulate a virtual laparoscopic hepatectomy. This simulation is based on a patient data set[4]). During such a procedure, multiple contacts occur between the liver and the ribs, the stomach, and the colon, among others (see figure 9) and a part of the liver (containing the tumour) is removed. This resection is performed progressively (after clamping the major veins located in the section to be removed) using an electric cautery device, a harmonic scalpel, or similar devices.

The scene is simulated using a time step of 40 msec for the Euler integration scheme. The laparoscopic grasper is modelled using articulated rigid bodies and the model is connected with Xitact IHP haptic interface. Ribs are considered as rigid, while the liver, stomach and colon are modelled as soft tissues using the co-rotational implicit FEM model. Contacts are detected and sampled using a Layered Depth Images (LDI) approach (Faure et al., 2008; Allard et al., 2010). During interactions without cutting, computation times are consistently below 35 msec for the whole simulation, and using multi-rate technique, haptic forces are updated at about $1kHz$.

By introducing dry friction through Coulomb's law in our contact model, we obtain realistic grasping of the soft-tissues: if not tightly closed, the grasper can slip along the tissue surface. Through the proposed approach, different collision forces can be rendered with high-fidelity: the user can feel the force when grasping the liver but also the change of this force when the liver collides with the ribs, the stomach and the colon. Moreover, the user can feel the stick-slip transition when the grasper releases the liver. Interactive cutting is also possible (see Figure 9), but as mentioned previously, remains currently limited to meshes of moderate size.

[4]Courtesy of IRCAD, www.ircad.fr/softwares/3Dircadb/3Dircadb2

6. Conclusions

In this paper, we have presented new methods for simulating in real-time realistic soft-tissues deformations, cutting and complex contacts with haptics. All these contributions rely on a co-rotational implicit FEM formulation and efficient GPU parallelizations. The results, demonstrated on a patient-specific laparoscopic hepatectomy training system, represent an important step toward simulation-based planning of complex procedures. While our results show the important speed-up that can be obtained using GPUs, optimal performance are not always straightforward to obtain.

Validation techniques are not presented in this paper. However, reader can refer to Marchal et al. (2008) which compare the co-rotational method with analytical solution. The relative energy error represent less than 2% in case of large rotations. Then, we consider the fast co-rotational model sufficiently accurate to be used in an interactive simulator.

On the other hand, we also used the compliance warping approximation to solve contacts. But, this approximation concerns only the deformation caused by contacts and the corrective motion associated. Moreover, the approximation is partly corrected by the free motion, based on exact constitutive law, of the next time-step.

Finally, the element subdivision can produce an ill-conditioned element which can cause numerical instability. Actually, this problem has not been completely solved so far and still remains an open problem of cutting simulation.

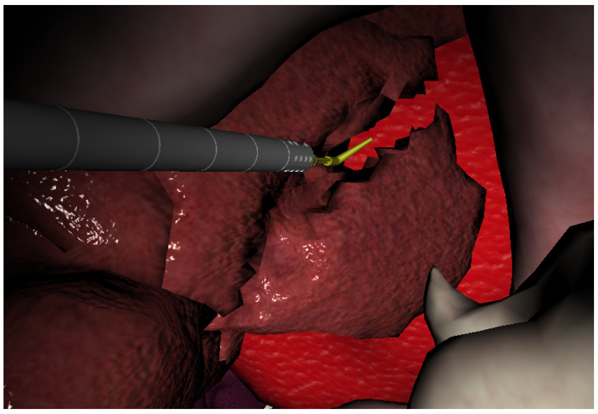


Figure 9: *Real-time simulation of laparoscopic hepatectomy. Realistic soft tissue deformations, cutting and complex contacts with haptics are possible using a combination of novel methods based on a co-rotational FEM formulation and dedicated GPU implementations.*

Bibliography

Acary, V., Brogliato, B., 2008. Numerical methods for non-smooth dynamical systems: Applications in mechanics and electronics. Lecture Notes in Applied and Computational Mechanics 35.

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). pp. 1–6, http://sofa-framework.org.

Allard, J., Faure, F., Courtecuisse, H., Falipou, F., Duriez, C., Kry, P. G., 2010. Volume contact constraints at arbitrary resolution. ACM Trans. Graph. (SIGGRAPH 2010) 29 (4), 1–10.

Anitescu, M., Potra, F., Stewart, D., 1999. Time-stepping for three-dimentional rigid body dynamics. Computer Methods in Applied Mechanics and Engineering (177), 183–197.

Baraff, D., Witkin, A., 1998. Large steps in cloth simulation. In: Proc. of ACM SIGGRAPH 98. pp. 43–54.

Barbič, J., James, D. L., 2005. Real-time subspace integration for st. venant-kirchhoff deformable models. ACM Trans. Graph. (SIGGRAPH 2005) 24 (3), 982–990.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., der Vorst, H. V., 1994. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition. SIAM, Philadelphia, PA.

Bro-Nielsen, M., Cotin, S., 1996. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In: Computer Graphics Forum. Vol. 15. pp. 57–66.

Brown, J., Sorkin, S., Latombe, J.-C., Montgomery, K., Stephanides, M., 2002. Algorithmic tools for real-time microsurgery simulation. Medical image analysis 6 (3), 289–300.

Comas, O., Taylor, Z., Allard, J., Ourselin, S., Cotin, S., Passenger, J., Jul. 2008. Efficient nonlinear fem for soft tissue modelling and its gpu implementation within the open source framework sofa. In: Proceedings of ISBMS 2008.

Cotin, S., Delingette, H., Ayache, N., 1999. Real-time elastic deformations of soft tissues for surgery simulation. IEEE Transactions on Visualization and Computer Graphics 5 (1), 62–73.

Courtecuisse, H., Allard, J., 2009. Parallel dense gauss-seidel algorithm on many-core processors. In: Proc. IEEE Int. Conf. on High Performance Computing and Communications. pp. 139–147.

Delingette, H., Jul. 2008. Biquadratic and quadratic springs for modeling st venant kirchhoff materials. In: Proceedings of ISBMS 2008. pp. 40–48.

Duriez, C., Dubois, F., Kheddar, A., Andriot, C., 2006. Realistic haptic rendering of interacting deformable objects in virtual environments. IEEE TVCG 12 (1), 36–47.

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).

Felippa, C. A., 2000. A systematic approach to the element independent corotational dynamics of finite elements. Tech. Rep. CU-CAS-00-03, Center for Aerospace Structures.

Forest, C., Delingette, H., Ayache, N., June 2004. Surface contact and reaction force models for laparoscopic simulation. In: International Symposium on Medical Simulation.

Grantcharov, T., Kristianson, V., Bendix, J., Bardram, L., Rosenerg, J., Funch-Jensen, P., 2004. Randomized clinical trial of virtual reality simulation for laparoscopic skills training. Br J Surg. 91, 146–150.

Harders, M., 2008. Haptic Rendering: Algorithms and Applications. A K Peters, Ltd. M.Lin et M.Otaduy, Ch. Haptics in Medical Applications, pp. 589–612.

James, D., Pai, D., 1999. Artdefo: Accurate real time deformable objects. In: 26th International Conference on Computer Graphics and Interactive Techniques. Proceedings of SIGGRAPH, ACM. pp. 65–72.

Johnson, A., 1985. Contact Mechanics. Cambridge University Press.

Joldes, G. R., Wittek, A., Couton, M., Warfield, S. K., Miller, K., 2009a. Real-time prediction of brain shift using nonlinear finite element algorithms. In: Proceedings of MICCAI 2009. Springer-Verlag, Berlin, Heidelberg, pp. 300–307.

Joldes, G. R., Wittek, A., Miller, K., 2009b. Suite of finite element algorithms for accurate computation of soft tissue deformation for surgical simulation. Medical Image Analysis 13 (6), 912 – 919, includes Special Section on Computational Biomechanics for Medicine.

Jourdan, F., Alart, P., Jean, M., 1998. A gauss-seidel like algorithm to solve frictional contact problems. Comp. Meth. in Appl. Mech. and Engin., 33–47.

Lee, B., Popescu, D. C., Joshi, B., Ourselin, S., 2005. Efficient topology modification and deformation for finite element models using condensation. In: Medicine Meets Virtual Reality. pp. 299–304.

Mahvash, M., Hayward, V., 2004. High-fidelity haptic synthesis of contact with deformable bodies. IEEE Comput. Graph. Appl. 24 (2), 48–55.

Marchal, M., Allard, J., Duriez, C., Cotin, S., Jul. 2008. Towards a framework for assessing deformable models in medical simulation. In: Proceedings of ISBMS 2008. Springer, pp. 176–184.

Marescaux, J., Clement, J.-M., Tassetti, V., Koehl, C., Cotin, S., Russier, Y., Mutter, D., Delingette, H., Ayache, N., 1998. Virtual reality applied to hepatic surgery simulation : The next revolution. Annals of Surgery 228 (5), 627–634.

Montgomery, K., Bruyns, C., Brown, J., Thonier, G., Tellier, A., Latombe, J.-C., 2002. Spring: A general framework for collaborative, real-time surgical simulation. In: Medicine Meets Virtual Reality (MMVR02).

Mor, A. B., Kanade, T., 2000. Modifying soft tissue models: Progressive cutting with minimal new element creation. In: Proceedings of MICCAI 2000. pp. 598–607.

Munshi, A. (Ed.), 2008. The OpenCL Specification Version: 1.0. The Khronos Group.

Murty, K., 1997. Linear Complementarity, Linear and Nonlinear Programming. Internet Edition.

Nickolls, J., Buck, I., Garland, M., Skadron, K., 2008. Scalable parallel programming with CUDA. ACM Queue 6 (2), 40–53.

Peercy, M., Segal, M., Gerstmann, D., 2006. A performance-oriented data parallel virtual machine for GPUs. In: ACM SIGGRAPH Sketches. p. 184.

Picinbono, G., Delingette, H., Delingette, H., Ayache, N., Ayache, N., 2000. Non-linear anisotropic elasticity for real-time surgery simulation. Graphical Models 65, 305–321.

Saupin, G., Duriez, C., Cotin, S., Jul. 2008. Contact model for haptic medical simulations. In: Proceedings of ISBMS 2008. pp. 157–165.

Seymour, N., Gallagher, A., O'Brien, M., Roman, S., Andersen, D., Satava, R., 2002. Virtual reality training improves operating room performance: results of a randomized, double-blinded study. Ann Surg. 236 (3), 458–464.

Sorensen, T., Greil, G., Hansen, O., Mosegaard, J., 2006. Surgical simulation–a new tool to evaluate surgical incisions in congenital heart disease? Interactive cardiovascular and thoracic surgery 5 (5), 536–539.

Taylor, Z., Comas, O., Cheng, M., Passenger, J., Hawkes, D., Atkinson, D., Ourselin, S., Sep. 2008. Modelling anisotropic viscoelasticity for real-time soft tissue simulation. In: Proceedings of MICCAI 2008. pp. 703–710.

Wriggers, P., 2002. Computational Contact Mechanics. Wiley ed.