An Intestinal Surgery Simulator: Real-Time Collision Processing and Visualization

Laks Raghupathi†, Laurent Grisoni‡, François Faure†
Damien Marchal‡, Marie-Paule Cani† and Christophe Chaillou‡

***Abstract*— This research work is aimed towards the development of a VR-based trainer for colon cancer removal, that enables the surgeons to interactively view and manipulate the concerned virtual organs as during a real surgery. First, we present a method for animating the small intestine and the mesentery (the tissue that connects it to the main vessels) in real-time, thus enabling user-interaction through virtual surgical tools during the simulation. We present a stochastic approach for fast collision detection in highly deformable, self-colliding objects. A simple and efficient response to collisions is also introduced in order to reduce the overall animation complexity. Second, we describe a new method based on generalized cylinders for fast rendering of the intestine. An efficient curvature detection method, along with an adaptive sampling algorithm is presented. This approach, while providing improved tessellation without the classical self-intersection problem, also allows for high-performance rendering, thanks to the new 3D skinning feature available in recent GPUs. The rendering algorithm is also designed to ensure a guaranteed frame rate. Finally, we present the quantitative results of the simulations and describe the qualitative feedback obtained from the surgeons.**

***Index Terms*— virtual reality, physically-based modeling, animation, curve and surface representation**

I. Introduction

Minimally invasive surgical (MIS) procedures are gaining popularity over open procedures among surgeons and patients. This is mainly due to lesser post-operative pain, fewer infections and an overall faster recovery. The tremendous success of laparoscopic cholecystectomy (gall bladder removal) has prompted surgical practitioners and educators to apply such techniques to other gastrointestinal procedures. In this paper, we focus on *laparoscopic colectomy* (colon cancer removal). Studies show that many patients undergoing this procedure benefit from the advantages of MIS procedures listed above while sharing the same risks of the corresponding open procedure [1]. Yet, as with most laparoscopic procedures, it is difficult to master with a very steep learning curve [2]. As part of the current training procedure, surgeons practice on pigs to get a feel of the organ's behavior. However, this technique is prohibitively expensive and also raises numerous ethical issues. We believe that a VR-based simulator platform can significantly help non-specialist surgeons and medical residents to acquire the necessary skills to perform this surgery in a cost-effective way. This may well result in popularizing the use of laparoscopic technique for this procedure thus benefiting more patients. Thus, our aim is to simulate the behavior of the intestine when the surgeon is practicing in the virtual surgical environment. Note that the current scope of this research work does not include the simulation of the removal of the cancer itself. But we focus on two important problems associated with training: (1) *Camera positioning* by allowing the trainee to visualize the relevant organs in 3D, (2) *Manual dexterity* by letting them interactively manipulate these organs. For many surgeons who are trained primarily in open techniques, this may help to overcome the perceptual and motor challenges associated with MIS procedures.

We will first review the background of the problem and highlight the challenges they pose. During this surgical procedure, the patient is lying on his back. As a result, the small intestine (henceforth simply referred to as *intestine*) is positioned just above the colon region, hiding the colon beneath (Fig. 1). The intestinal region of a human body is characterized by a very complex anatomy. The intestine is a tubular structure, about 4 m long, constrained within a small space of the abdominal cavity, resulting in the creation of numerous *folds*. This is further complicated by a tissue known as the *mesentery* that connects the intestine to the blood vessels [3] (Fig. 2). An important surgical task of this procedure is to displace the tissues and organs away by pulling and folding them from the site of the operation [4]. As the surgeon manipulates these organs, they deform and collide with each other. Thus the broad challenges here

Manuscript received MMDDYY; revised MMDDYY.
†GRAVIR/IMAG, Grenoble, France
‡LIFL, University of Lille 1, Villeneuve d'Ascq, France
Corresponding author's email: laks@imag.fr

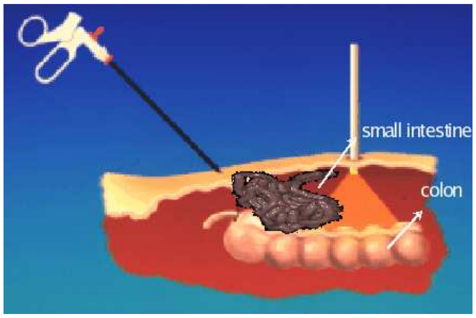


Fig. 1. Position of the intestine during the surgery

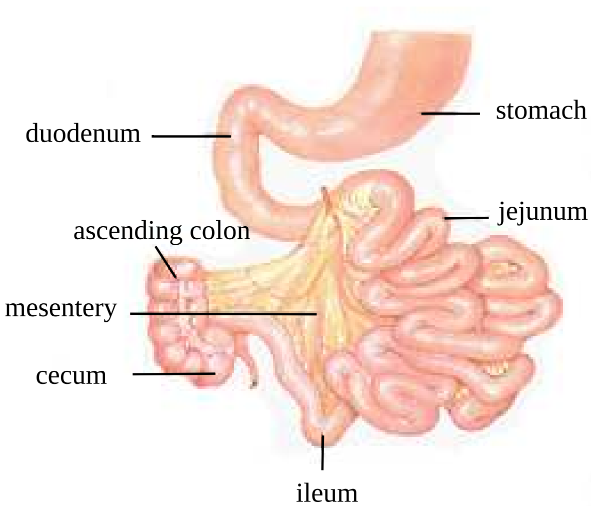


Fig. 2. Anatomy of the intestinal region

are the real-time animation and visualization of these organs at an acceptable frame rate (a minimum of 25 frames a second). Our overall approach to solving this problem consists of a layered model: a skeletal axis deformed using physical-based animation, rendered with a generalized cylinder-based skinning. Thus in order to animate these organs in real-time, we propose to:

- efficiently model the intestine and the mesentery taking into account its geometry
- detect the collisions and self-collisions occurring in the intestinal region during animation
- provide a fast and stable collision response

The skeletal model used for animation should be covered with a triangulated mesh which can be realistically shaded or textured. However, a naïve skinning approach will create tessellation problems in high-curvature regions. Hence we present:

- an efficient method to detect high-curvature regions and to adapt the sampling of the axis accordingly
- an algorithm that dynamically changes the skinning parameters to maintain a pre-imposed frame rate
- a hardware-based skinning feature for fast and smooth rendering available in recent GPUs

All these contributions have been implemented within a surgical simulator platform [5] that can be easily evaluated by the surgeons. Note that the first versions of our contribution on intestine animation and rendering appeared as separate conference proceedings [6], [7]. This paper gives an extended presentation of both, describes their integration into a simulator platform, provides the results of the integration and summarizes the evaluation of the results, both qualitative and quantitative.

The remainder of this paper organized as follows: In Section II, we briefly describe the previous work related to our problem, both from animation and rendering perspectives. We then elaborate on creating a virtual intestine and mesentery, an initial geometric model defining their shape and a mechanical model for animation in Section III. Section IV presents our collision detection and response algorithms in detail. We then propose a method for fast rendering, along with a self-tuning algorithm for guaranteed frame rate in Section V. Section VI briefly describes the integration into a surgical simulator platform. The results and validation are discussed in Section VII followed by the conclusions in Section VIII.

II. Previous Work

A. Real-time Animation of Deformable Models

Recently, many researchers have focused on the efficient simulation of deformable models with some of them using adaptive, multi-resolution techniques [8], [9], [10] and some applied to surgery simulators [11], [12], [13], [14], [15], [16]. In all these works, volumetric deformable bodies were either simulated in isolation, or were interacting with a single rigid tool, enabling the use of very specific techniques for collision detection and response, such as methods based on graphics hardware [17]. The problem we have to solve here is different: as will be shown in the next section, no volumetric deformable model will be needed since the intestine and the mesentery can be represented as a 1D and a 2D structure respectively. Moreover, the main issue is detecting and processing the collisions and self-collisions of the intestinal system in real-time. Accordingly, a simple spline model animated by mass-spring dynamics was used by France [18], [19] for simulating the intestine. France used a grid-based approach for detecting self-collisions of the intestine and collisions with its environment. All objects were first approximated by bounding spheres, whose positions were stored, at each time step, in the 3D grid. Each time a sphere was inserted into a non-empty voxel, new collisions were checked within this voxel. Though this method achieved

real-time performances when the intestine alone was used, it did not handle the simulation of the mesentery.

A well-known technique for accelerating collision detection is to approximate the objects by a bounding volume hierarchy (BVH) [20], [21], [22], [23], [24], [25]. Such approaches provide simple tests for fast detection of non-colliding regions. The BVH can be recursively updated when the objects undergo small deformations. However, this is not suitable for intestine-mesentery interaction where, even a small local deformation can potentially cause a large movement of the folds. This creates a *global deformation* at large scale, which prevents the BVH from being efficiently updated. An alternate multi-resolution method, based on layered shells, was recently presented by Debunne [26]. It is well-suited for collision detection between deformable objects since the shells themselves are deformable structures extracted from a multi-resolution representation of these objects. Though suitable for volumetric deformable bodies, this method will not be appropriate for intestine and mesentery, since the time-varying folds cannot easily be approximated at a coarse scale. Debunne also exploited temporal coherence following Lin and Canny's [27] idea of detecting collisions between convex polyhedra by tracking pairs of closest vertices. These pairs were efficiently updated at each time-step by propagating closest distance tests from a vertex to its neighbors. Debunne adapted this technique for detecting collisions between his volumetric layered shells very efficiently. Since these shells were neither convex nor rigid, a stochastic approach was used at each time step to generate new pairs of points anywhere on the two approaching objects. These pairs were made to converge to local minima of the distance, disappearing when they reached an already detected minimum. Our work is inspired by this idea of *stochastic* collision detection exploiting temporal coherence. It has been adapted here to the specific processing of multiple collisions and contacts between the intestine and the mesentery folds.

B. Intestine and Mesentery Rendering

In addition to handling the collisions, we also need to provide a real-time, smooth rendering of the intestinal folds. The intestine and the mesentery are geometrically complex organs and also topologically distinct from each other: a deformable cylinder-like object and a deformable, non-developable tissue. As a result, it is natural to study their rendering separately. Many efficient techniques do exist for surfaces (a simple tessellation actually gives good results). Hence, we shall devote more preference to deal with the much harder problem of intestine rendering.

A possible approach to intestine rendering is to use implicit representation [18]. This is indeed a powerful representation, especially for handling topologically complex or deformable objects. Yet, it is not well-suited for our purpose, since none of its classical features are useful here. Moreover, the blending property inherent to implicit surface implies that some control operations would have to be performed on the implicit intestine model. This additional processing is necessary in order to avoid unwanted blending between the intestinal folds.

A more natural approach to intestine modeling is to use generalized cylinders, also sometimes called *sweep*. To our knowledge, only very few previous techniques dealt with the efficient rendering of such objects. Classical generalized cylinders are defined using a parameterized axis $C(u)$, along with a set of planar cross-section that can be represented as a continuous set $S(u)$ depending on the same u scalar value that parameterizes the axis [28]. The classical tessellation approach is as follows: the axis C is sampled either uniformly or adaptively, depending on some curvature isolation (see [7] for an extensive overview of the possible techniques). For each axis sample, the section S is positioned on a local frame calculated along the axis [29], and then approximated by a poly-line. Two consecutive poly-line sections along the axis are then connected by a triangle stripset in order to form a local approximation of the sweep. High curvature point detection becomes a critical step for such a process and any sharp angle is likely to be missed in the sampling. It is also likely that the tessellated sections can overlap, hence providing the user with a self-intersecting mesh (see Fig. 3).

Our solution rather comes from skinning and thus takes advantage of the OpenGL skinning feature provided by recent GPUs [30]. Skinning (also called *vertex blending*) is most commonly used in character deformation applications that allow continuous deformation of a skin mesh over an animated skeleton. The principle of this extension is as follows: it is a standard practice to define a simple geometric transformation in homogeneous coordinates by a 4x4 matrix. This composition of rotation, translation and scaling allows an initial mesh to be positioned anywhere within a given scene along with simple deformations. For an object, skinning uses two matrices (i.e., two different positions of the object), and for all the vertices of the object, a scalar value (called *weight*) is used to interpolate the vertex position. Accordingly, given an initial vertex ν^0, with weight $\omega(\nu^0)$ and two transformations M_1 and M_2, the resulting position ν of the vertex is given by:

$$\nu = \omega(\nu^0)M_1\nu^0 + [1 - \omega(\nu^0)]M_2\nu^0 \qquad (1)$$

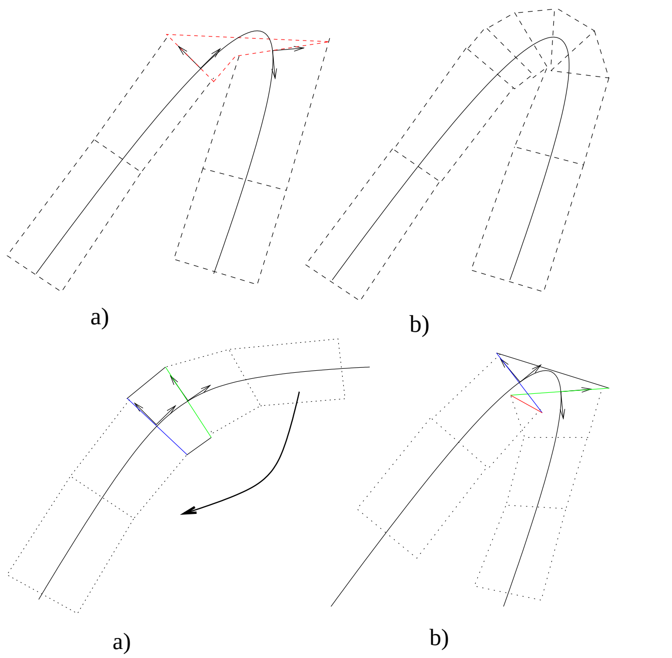


Fig. 3. Classical tessellation problems on generalized cylinders. Top: Inadequate sweep tessellation (left) and correct one (right). Bottom: Mesh self-intersection when deforming the initial sweep.

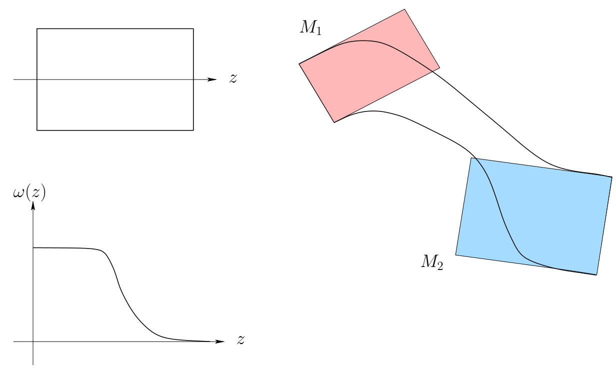


Fig. 4. Simple skinning application to a polygonal primitive. Combination of M_1 and M_2 transformations on the object (top-left), using the weights (bottom-left), gives the results (right).

The above transformation can be done entirely by the GPU with no CPU computation involved. Setting optimal weights for a given deformation is, in general, a difficult problem. Recently, Bloomenthal presented a technique for automatically setting weights when skinning is used in character animation [31].

The above transformation process allows for simple deformation of the initial mesh. Fig. 4 shows a simple 2D example of what is obtained using initial rectangle deformation, using weights that only depend on the z position along the rectangle axis. In this figure, the weight function is similar to the blending functions described for other purposes in [32]. Controlling the weight distribution curve determines the way the GPU will interpolate from one transformation to the other. Indeed, this technique cannot be directly applied to the rendering of a folded generalized cylinder controlled by a skeleton, the intestine in our case. Extending the skinning method to this case will be the main contribution of Section V.

III. Modeling and Animating the System

Our overall aim is not to develop an accurate, patient-specific trainer but rather a generic simulator which can help the surgeons to practice the gestures used to manipulate the organs. The first problem we have to solve is to create a virtual model of the intestine and the mesentery which should also be the basis for the animation and rendering stages. In this section, we describe the actual anatomy in brief, provide the basis of our approach and present the details of our model. We further present the mechanical model needed for animating the intestinal system.

A. Anatomical Model

In order to extract the anatomy of the intestine and the mesentery, we sought the help of our medical collaborator IRCAD in Strasbourg (a digestive cancer research institute [33]). With the current imagery techniques, they found it impossible to extract the complex anatomy the organs such as the mesentery. Hence we decided to come up with something simpler than the actual geometry, but yet can capture the overall behavior.

The mesentery is a folded surface membrane, approximately 15 cm wide, which links the intestine, a long tubular structure, to the main vessels of 10 cm length (Fig. 2). Since the mesentery cannot be unfolded onto a plane, setting up its initial geometry free of self-intersections is quite difficult. We solved the problem by modeling a possible rest position for the intestine as a folded sine curve lying on the inner surface of a cylinder of radius 15 cm. The axis of the cylinder, 10 cm in length, represents the main vessels. The folds are placed on the cylinder such that their total length is 4 m (Fig. 5). Then the mesentery can be defined as the surface generated by the set of non-intersecting line segments linking the cylinder axis to the curve. Though this initial geometry is too symmetric to be realistic, this model gives adequate local geometric properties to the mesentery membrane (Fig. 6). When animated under the effect of gravity, this collision-free initial positions will automatically move to their correct positions. The geometry of the intestine is defined by creating a piecewise

tubular surface of radius 2 cm along its skeleton curve. The thickness of the mesentery surface, parameterized based on patient-specific data, was set to 1 cm.

B. Mechanical Considerations for Animation

For animation, our motivation to use mass-spring approach was derived from the following arguments:

- Mass-spring method is more effi cient and suitable over FEM for deforming bodies with large displacements and local elastic deformations
- In addition, an organ such as the intestine can have an infi nite number of rest states, whereas FEM is based on the notion of displacements w.r.t. a unique rest state thus making it unsuitable
- Mass-spring can provide stable simulation of deformable objects at moderate time-steps
- We can adjust the behavior of the system intuitively by adjusting the damping, stiffness, etc.
- Collision detection here is a much more complex task requiring more CPU-time over animation. So, a very complex mechanical model might slow down the simulation
- Finally, the *perceived quality* of most interactive 3D applications does not depend so much on exact simulation but rather on real-time response to collisions [34]

Accordingly, we designed a model consisting of mass points connected by damped springs. Since the mesentery has a much larger length (4 m near the intestine) than width (15 cm near the vessel), we sampled our model by four sets of 100 masses each (Fig. 6). The last set of masses requires no computation since they are attached to the main vessels, requiring only 300 masses to be integrated at each time step. In addition, no specific model is needed for the intestine since it can be simulated by adjusting the mass and stiffness values along the first bordering curve of the mesentery surface.

IV. Real-Time Collision Processing

A major computational bottleneck in many animation/simulation systems is the handling of collisions between the objects under the influence of external forces (gravity, user-input, etc.). In our case of real-time simulation, we need to perform this as quickly as possible. Failure to detect the collisions will result in interpenetration - an unrealistic behavior. In this section, we describe our approach, which efficiently detects the colliding regions and a response algorithm which prevents/corrects the interpenetrations.

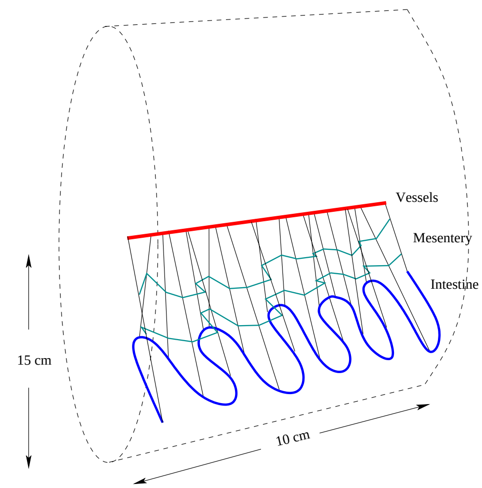


Fig. 5. Initialization of the geometry of the intestine and mesentery. The intestine is drawn on the inner surface of a small cylinder. Figure greatly simplifi ed for clarity.

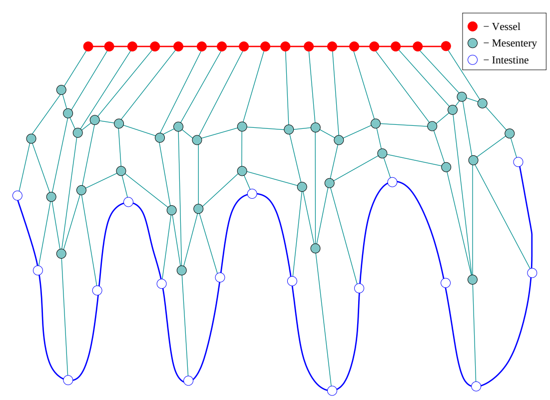


Fig. 6. Mechanical model of the organs (shown unfolded).

A. Collision Detection

Our method for real-time collision detection exploits *temporal coherence*, i.e., pairs of closest points between colliding bodies are tracked [26], [27]. The main differences here are: (1) the interacting objects have a tubular (intestine) or a membrane-like structure (mesentery), and (2) most collisions will be self-collisions between different folds of the same body. We first explain the collision detection method for the intestine alone, and then explain the mesentery case.

Collision detection between cylinders can be processed by computing the closest distance between their axes [35], and comparing them to the sum of their radii. For the intestine, computing the distance between two segments is done by considering the distance between their principal axes (A segment here refers to a simple line segment with it's end-points parameterized by $(s, t) \in [0, 1]$). We then store the (s, t) value corresponding to the closest point within the segments and the actual minimum distance $d_{\min}$.

Adapting the notion of "closest element pairs" to this skeleton curve means that we track the local minima of the distance between non-neighboring segments along the curve (Fig. 7). Of course, only the local minima satisfying a given distance threshold are considered relevant. We refer these pairs of segments as *active pairs*. Each active pair is locally updated at each time step, in order to track the local minima, when the intestine folds move. This is done by checking whether it is the current segment pair or a pair formed using one of their neighbors which now corresponds to the smallest distance. This update requires nine distance tests (Fig. 8), and the pair of segments associated to the closest distance becomes the new active pair. When two initially distant active pairs converge to the same local minimum, one of them is suppressed. A pair is also suppressed if the associated distance is greater than a given threshold.

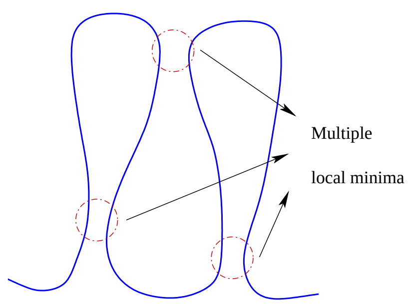


Fig. 7. Tracking the local minima of distance between non-neighboring segments

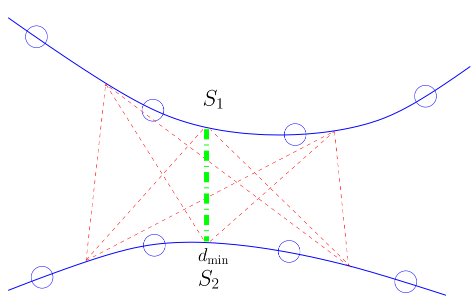


Fig. 8. Distance computation between two intestine segments

The above process tracks the existing regions of interest but does not detect new ones. Since the animation of the intestine may create new collisions, a method for creating new active pairs of segments is needed. Our approach is inspired from the stochastic approach of [26]. At each time step, in addition to the update of the currently active pairs, n additional random pairs of segments, uniformly distributed between the end-points, but under the distance threshold are generated. The update of these extra active pairs is similar to the update of the existing local minima. The complexity of the detection process thus linearly varies with the user-defined parameter n. At each time step, collision detection consists of selecting among the currently active pairs, the pairs of segments which are closer that the sum of their radii. Collision response (see IV-B), will then be applied between these segments.

For the mesentery, the total number of segments to be considered during each time-step is too large for real-time computation. We use the following approximations to reduce the complexity of the problem. Firstly, since the mesentery is very thin and soft compared to the intestine, its self-collisions will almost have no effect on the overall behavior of the system. Hence, we ignore the testing of these and only consider the non-trivial cases of intestine-intestine and intestine-mesentery.

Secondly, we use a two-step convergence algorithm to reduce the number of distance computation required for the mesentery. Accordingly, given a segment pair (S_1, S_2), we first find if there exists another segment $S_1^{'}$ that is the closest to S_2. ($S_1^{'}$ is S_1 if all other neighbors are farther to S_2). Then, we find a segment $S_2^{'}$ that is the closest to $S_1^{'}$. This update requires 13 distance computations at most (i.e., when one segment belongs to the intestine, and the other to the inside of the mesentery). When a collision is detected, we apply response force not only to the pair of deepest penetrating segment-pair but also to the entire collision area (both for intestine and mesentery). A recursive algorithm searches the neighbors to find all the colliding pairs in the area.

B. *Collision Response*

We initiate the response whenever the distance between the two segments is less than the sum of their radii. The earlier approaches such as the penalty method [36] and the reaction constraint method [37] implemented collision response by altering the force matrix in the mass-spring method. This force has to be of large magnitude in order to be effective in large time-step scenarios. However, this may cause segment displacements several times larger than their thickness thus creating new collisions and instabilities. Instead, our new method alters the displacements and velocities such that it instantaneously cancels the interpenetration while keeping a resting contact between the two colliding bodies with no bouncing effects.

Let the end-point velocities of segment S_1 be $\mathbf{v}_1$ and $\mathbf{v}_1^{'}$ and that of segment S_2 be $\mathbf{v}_2$ and $\mathbf{v}_2^{'}$ respectively. Let $\mathbf{x}_1$, $\mathbf{x}_1^{'}$, $\mathbf{x}_2$ and $\mathbf{x}_2^{'}$ be the corresponding end-point positions. Let $\mathbf{v}_{c1}$ and $\mathbf{v}_{c2}$ be the velocities of the closest approaching point within each segment and $\mathbf{x}_{c1}$ and $\mathbf{x}_{c2}$

be the positions of the closest points (Fig. 9). Let $\bar{s} = 1 - s$ and $\bar{t} = 1 - t$. We have:

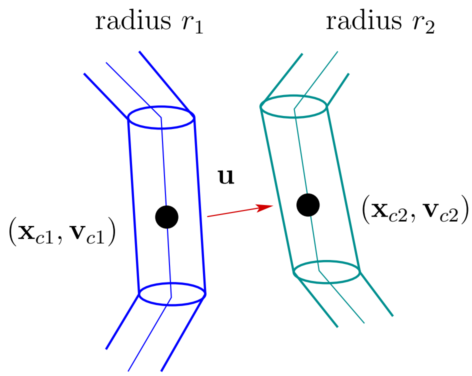


Fig. 9. Collision response by displacement-velocity correction

$$\mathbf{v}_{c1} = \bar{s}\mathbf{v}_1 + s\mathbf{v}_1^{'} \qquad \mathbf{v}_{c2} = \bar{t}\mathbf{v}_2 + t\mathbf{v}_2^{'} \tag{2}$$

Let two impulses per time-step, f and $f'(= -f)$ (one for each segment), be applied along the direction of collision $\mathbf{u}$ to cause a velocity change such that the relative velocities along $\mathbf{u}$ is zero. These impulses should set the new velocities $\mathbf{v}_{newc1}$ and $\mathbf{v}_{newc2}$ such that:

$$(\mathbf{v}_{newc1} - \mathbf{v}_{newc2}).\mathbf{u} = \mathbf{0} \tag{3}$$

This cancels the penetration velocity and avoids any bouncing. The impulse f acting on the point of collision can be split between the end-points according to their barycentric coordinates. Then we have:

$$\begin{aligned} \mathbf{v}_{new1} &= \mathbf{v}_1 + \frac{\bar{s}f}{m_1}\mathbf{u} \qquad & \mathbf{v}_{new1}^{'} &= \mathbf{v}_1^{'} + \frac{sf}{m_1^{'}}\mathbf{u} \\ \mathbf{v}_{new2} &= \mathbf{v}_2 + \frac{\bar{t}f}{m_2}\mathbf{u} \qquad & \mathbf{v}_{new2}^{'} &= \mathbf{v}_2^{'} + \frac{tf}{m_2^{'}}\mathbf{u} \end{aligned} \tag{4}$$

where m_1, $m_1^{'}$, m_2 and $m_2^{'}$ are the masses of the end-points. Again, expressing the new velocity of the colliding point $\mathbf{v}_{newc1}$ in terms of $\mathbf{v}_{new1}$ and $\mathbf{v}_{new1}^{'}$:

$$\begin{aligned} \mathbf{v}_{newc1} &= \bar{s}\mathbf{v}_{new1} + s\mathbf{v}_{new1}{'} \\ &= \mathbf{v}_{c1} + (\frac{\bar{s}^2}{m_1} + \frac{s^2}{m_1^{'}})f\mathbf{u} \end{aligned} \tag{5}$$

Similarly for segment S_2:

$$\mathbf{v}_{newc2} = \mathbf{v}_{c2} - (\frac{\bar{t}^2}{m_2} + \frac{t^2}{m_2^{'}})f\mathbf{u} \tag{6}$$

Substituting (5) and (6) into (3), we have:

$$f = \frac{(\mathbf{v}_{c2} - \mathbf{v}_{c1}).\mathbf{u}}{\frac{\bar{s}^2}{m_1} + \frac{s^2}{m_1^{'}} + \frac{\bar{t}^2}{m_2} + \frac{t^2}{m_2^{'}}} \tag{7}$$

Using this value of f, we compute the new velocities of the end-points from (4). We use a similar formulation for correcting the positions of colliding segments. The only difference is in the condition for avoiding interpenetration, which considers the segment radii r_1 and r_2:

$$(\mathbf{x}_{newc1} - \mathbf{x}_{newc2}).\mathbf{u} = r_1 + r_2 \tag{8}$$

The integrated impulse value g to change the positions in order to enforce the above condition is then:

$$g = \frac{(\mathbf{x}_{c1} - \mathbf{x}_{c2}).\mathbf{u} + r_1 + r_2}{\frac{\bar{s}^2}{m_1} + \frac{s^2}{m_1^{'}} + \frac{\bar{t}^2}{m_2} + \frac{t^2}{m_2^{'}}} \tag{9}$$

g is used to modify the end-point positions as in (4).

We note that updating position and velocity of one edge may still create or cancel collisions among other segments. A possible solution is to repeat the collision check for all the pairs in our list in order to identify such pairs. But this may result in an endless loop. So in our implementation, we prefer using a fixed number of iterations even if we miss some collisions during the current time-step. Handling multiple collisions in general a difficult problem even for rigid bodies with no straightforward solution. Some approaches based on a global solution (simultaneous handling of all the collisions) have been proposed. Giang et al. [38] constructed this as a constrained optimization problem and used a LCP-based solution. Needless to say, it is much more difficult for flexible bodies with real-time constraints and to our knowledge, no such techniques exist for our case.

V. Skinning and Rendering

In this section, we present our approach for intestine and mesentery rendering. We shall deal with them separately as explained before. Medical experience showed that it is very hard to perceive how the mesentery actually deforms. However we did observe that this surface most often lies behind the intestine during surgical manipulations. Thus the only main purpose of simulating the mesentery is to ensure a proper mechanical behavior of the intestine. As a result, we chose to use a very coarse rendering representation for the mesentery. The rendering primitive is simply a triangle stripset, whose points are the intestine mechanical sampling points and the mesentery root. Using two separate models for mechanics and rendering did not attract any negative comments from the surgeons during demonstration, which in a way vindicates our presumption. The following three subsections describe the approach we used for efficiently rendering the intestine.

A. High-Curvature Detection and Adaptive Sampling

We first convert the mechanical sampling points of the intestine axis, into a cardinal spline approximation

(an *interpolation* spline) - a straightforward process [39]. This is done in order to counterbalance the piecewise linearity of the mechanical model. The underlying idea behind the sampling process is fairly simple. Many interesting spline models have a regularity property [40]. This analytic property has among its consequences that no undesired oscillation appears in a given spline segment. In other words, a curve's overall shape can somewhat be pre-determined by studying its control point configuration. Although we used cardinal splines in our experiments, most of these techniques can easily be generalized to other cases [7].

The sampling process uses the control point sequence $(P_0, P_1, \ldots, P_n)$ of C, the corresponding key parameter values $(u_0, u_1, \ldots, u_n)$ (we chose uniform in our platform, i.e., $u_k = k\delta$, δ being a constant value), and some threshold value ε, as input variables. Recall that these control points are generated from the spline approximation procedure from the mechanical sampling points. The sampling algorithm is split into two steps:

Step 1: A first pass calculates the indicators of the local spline curvature. The spline regularity ensures that such a perturbation can be studied considering the control point configuration. A sequence of scalar values $\{c_i\}_{i=0\ldots n}$ is defined using:

$$c_i = \frac{1}{2}\left(\frac{\mathbf{P}_{i-1,i}.\mathbf{P}_{i,i+1}}{\|\mathbf{P}_{i-1,i}\|\|\mathbf{P}_{i,i+1}\|} + 1\right) \tag{10}$$

where $\mathbf{P}_{i,j} = P_j - P_i$. Extremal values c_0 and c_n are treated considering two additional ghost sampling points, created using Bessel technique [39], i.e., the ghost point at u_{-1} (similarly u_{n+1}) is calculated using the unique parabola that interpolates P_0, P_1, P_2 (similarly P_n, P_{n-1}, P_{n-2}). Each spline segment connection is therefore associated with a numerical value c_i that measures the local control point curvature. Fig. 10 shows a visual evaluation of this step, associating different color to each segment, depending on the numerical values of c_i. Here, the darker regions correspond to those segments that the algorithm selects as high curvature ones.

Step 2: A second pass further processes the segments that potentially contain high-curvature points (only one such point is possible per spline segment, once again thanks to regularity). For each parameter interval $[u_i, u_{i+1}]$, if $|c_i| < \varepsilon$ and $|c_{i+1}| < \varepsilon$, then the spline segment $[u_i, u_{i+1}]$ is treated as a whole by the display process. Otherwise, the corresponding spline segment potentially contains a high-curvature point which can be detected by a symbolic root extraction. In the case of uniform cardinal splines, the maximal curvature parameter point m_i on a segment satisfies a linear equation. Using this, the parameter segments that are actually drawn are

Fig. 10. Spline segment coloration to illustrate our curvature isolation process, during interactive sweep deformation

$[u_i, m_i - \delta_i]$, $[m_i - \delta_i, m_i + \delta'_i]$ and $[m_i + \delta'_i, u_{i+1}]$, with

$$\delta_i = \frac{m_i - u_i}{3} \quad \text{and} \quad \delta'_i = \frac{u_{i+1} - m_i}{3} \tag{11}$$

Such a separation ensures that the frame calculation (see V-B), is always done at low curvature regions of the spline ensuring numerical stability. Thus the regularity property ensures that high curvature segments are detected during the first pass of the process itself. It only involves a simple, fast computation and thus fits the requirement for real-time rendering. The second pass positions the additional sampling points in the high-curvature segments in order to get a more stable tessellation. This sampling process is somewhat related to the surface analysis technique described in [41], where both symbolic and numerical tools are combined for solving the addressed problem. Here, the high curvature segments are isolated using numerical evaluators c_i, and maximal curvature points are calculated using symbolic extraction on the considered spline segment. The main difference is the simplification done in order to achieve faster performance. However, with this modification (especially regarding c_i), no theoretical validation is available. In addition, our scheme does not guarantee the minimality of the number of samples with respect to any error criteria. It simply ensures that no high curvature point will be missed. Our method makes sure that a spline curve is turned into a set of interest points along the curve, i.e., those corresponding to the segment extremities and those belonging to the high-curvature region. The resulting curve segments are displayed using a hardware-based technique described in the following section.

B. Hardware-based Rendering using Skinning

The skinning definition (see Section II) provides us with quite a powerful tessellation primitive; more complex than an ordinary polygon. A *tessellation elementary*

cell will be defined from a regular cylinder with a standard weight function and two matrices M_1 and M_2. Combining this simple, pre-computed cylindrical shape with the skinning deformation process significantly reduces the software side of the tessellation process. The overall tessellation algorithm is done in two steps:

Step 1: The sampling process gives a set of parameter intervals $[a_j, a_{j+1}]$ ($\{a_j\}$ is a reordered set of $\{u_i\}$ with the inserted parameters of the high-curvature region).

Step 2: For each interval generated by the sampling algorithm, two frames (one for each segment extremity) are calculated along the axis C as follows [29]. Each frame (t_j, k_j, b_j) corresponding to the parameter value a_j is calculated from the previous frame using

$$b_j = t_j \times k_{j-1} \quad \text{and} \quad k_j = b_j \times t_j{}^1 \tag{12}$$

The t_j vectors are always assigned the first derivative of the local curve. This reduces the frame twist along the curve which is not the case when using a simple Frenet frame. However, for the first frame alone (corresponding to a_0), we use the Frenet frame. Each frame gives the local orientation of the sweep cross-section. These frames generate the homogeneous transformation matrices M_1 and M_2 for the considered parameter interval. They are then used by the blending technique (Section II) on the following elementary object: a non-deformed cylinder aligned along the z-axis in our implementation

$$\begin{cases} x^2 + y^2 = 1 \\ z \in [0, 1] \end{cases} \tag{13}$$

This non-deformed cylinder is approximated by polygons using a positive integer n as the *approximation resolution*. In our implementation we used 2^{n+1} sections for the cylinder, uniformly positioned along the cylinder axis. All the sections are discretized at a given, constant, resolution. The higher the n value, the smoother the interpolation between the frames represented by M_1 and M_2. Many weight functions are available for smooth deformation results (see [7] for possible choices). In our case, the weight function should be smooth, locally supported and monotonically decreasing from 1 to 0. Accordingly, we used the following:

$$\omega(\nu) = \begin{cases} 1 - 2z_\nu^2, & z_\nu^2 < \frac{1}{4} \\ \frac{(1-z_\nu^2)^2}{0.75+1.5z_\nu^2}, & z_\nu^2 \geq \frac{1}{4} \end{cases} \tag{14}$$

z_ν being the z-coordinate of the vertex ν on the non-deformed cylinder defined in (13). This weight function is a particular case of the more general function proposed in [32] for soft objects blending.

[1]where $\times$ denotes a vectorial cross product.

Fig. 11 shows an example of tessellation achieved using this technique (hence, the display primitive is a deformed cylinder). We can observe that the results of this new technique are visually more pleasing than the standard approach (brute-force tessellation with piecewise linear approximation) and the high-curvature points are handled in a much better way (see box in Fig. 11). The software part of this tessellation only involves the calculation of the geometric transformations and no vertex position is explicitly evaluated and transmitted to the GPU during the display process. The tessellation primitive for a given resolution n is constant over time, since only the matrix changes from one parameter interval to another. Thus it can be stored in a *displaylist* using *vertexArray* technology [30], to take full advantage of the data transmission optimizations of the GPU. Note that

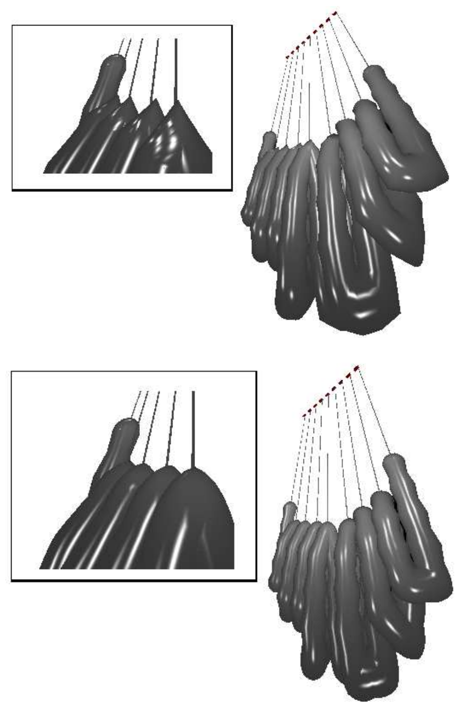


Fig. 11. Standard tessellation result (top), and high quality rendering using our new technique (bottom)

there is no mathematical proof that such a deformation tool can properly handle all the classical tessellation problems. We also observe that increasing the number of vertices on the tessellation primitive results in smoother deformation of it (Fig. 12). Though this may seem quite obvious, the smoothing involves only a small CPU overhead, since the deformation process is totally performed by the GPU. The only overhead needed is the transmission of the tessellation primitive to the GPU.

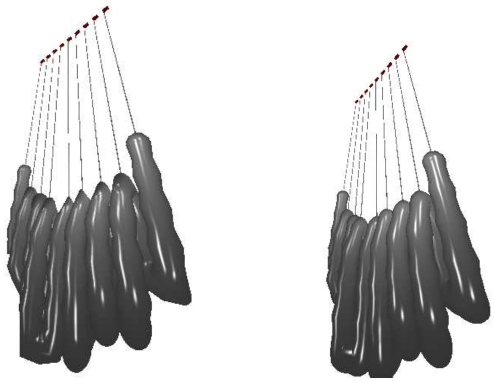


Fig. 12. Low quality tessellation (top) and high quality tessellation (bottom) obtained using our automatic frame rate adaptation system at 900 and 460 frames/sec respectively, on a GeForce 4 based system

C. Guaranteed Frame Rate

The accuracy of the entire visualization process described above depends on two parameters. The first parameter ε, determines how many parameter intervals will be used in the tessellation of the skeleton curve (this number should at least be equal to the number of spline segments on C). The second parameter n defines the complexity of the primitive cylinder used for tessellation. In our implementation, we actually use several versions of the tessellation primitive. The number of polygons in each version is $O(2^i)$, $i = 0, \ldots, n$ (section V-B). For a given parameter interval, increasing the number of vertices on the tessellation primitive produces smoother interpolation with only a small measured computational overhead. The tessellation system is combined with an algorithm that compares the current frame rate to a reference frame rate, and adapts the value of ε and n automatically. The principle of this algorithm can be seen as a feedback control loop. We tune the values of ε and n by a function of the difference between the measured frame rate and the required value. More precisely, the iterations from one display configuration (ε_k, n_k) to the next one $(\varepsilon_{k+1}, n_{k+1})$ use the following relations:

$$\begin{aligned} \varepsilon_{k+1} &= \varepsilon_k + \alpha(f_0 - f_k) \\ \rho_{k+1} &= \rho_k + \beta(\varepsilon_{k+1} - \varepsilon_k) \\ n_{k+1} &= \lfloor \rho_{k+1} \rfloor \end{aligned} \tag{15}$$

where ρ_k is a floating point version of the integer variable n_k, α and β are two arbitrary constants, f_k is the current measured frame rate, and f_0 the desired one. In our implementation, we used $\alpha = 10^{-4}$ and $\beta = 5$. These values were experimentally determined and proved to be satisfactory in practice. Fig. 12 illustrates the results. The tessellation process stabilizes itself at the desired frame rate in less than a second.

VI. System Integration

The models presented so far have been integrated in a simulation platform, for seamless interoperability with existing models and tools. We now briefly describe the SPORE library (see [5], [15] for more details). Developed in C++, this library provides a fully functional environment for the simulation of minimally invasive ovarian surgery. All the objects within this library are decomposed into three different models: a geometric model used for rendering, a mechanical model used for animation and a collision detection model used for evaluating the influence of the environment. The three layers of an object share the necessary information in order to access them (e.g., external forces are transferred from the collision model to the mechanical model which in turns transfers the state variables to the geometric model for rendering). The library is designed such that all the interactions between objects are sphere-based, which is a good trade off between speed and being generic. As a result, the system composed of the intestine and the mesentery has been included within the simulator as a whole and is treated by other simulated objects as a single entity. SPORE is organized as follows: a global animation process iteratively processes the user interactions and collisions between objects, in order to calculate the forces that will be applied to each object. It then gives an object the possibility to evaluate its own behavior, by calling an ad-hoc method of the object class, transferring the applied external forces to itself. This way, the system composed of intestine and mesentery can calculate the forces due to self-collision by itself using the previously described method, and its response to the global force system, using the presented animation technique. With this system integration, we can evaluate the simulator for its performance and realism, the results of which are presented in the following section.

VII. Results and Validation

Simulator Results: Snapshots from the real-time animation of the simulator are depicted in Fig. 13. In all cases, the organs were subject to forces due to gravity, user-input and collisions. Fig. 13(a) shows the case of an isolated intestine and mesentery manipulated by a virtual probe (the tiny sphere). Note that our displacement-velocity method or collision response produces fairly stable simulations (Fig. 13(b)) with some undesired vibrations. Figures 13(c) and (d) show the simulations inside a simulated abdominal cavity. All the snapshots were captured from our simulator in real-time in a standard PC with Bi-Athlon 1.2 GHz, 512 MB RAM and *n*VIDIA GeForce 3 graphics

card. A dynamic real-time demonstration of our simulator is available as a companion to this paper at: `http://www-imagis.imag.fr/Membres/Laks.Raghupathi/intestine.html`.

Quantitative Validation: The mechanical and collision detection model was initially implemented and tested independently prior to the integration. The results were fast enough to run at 30 Hz on a standard PC and the overall behavior was good. However, due to the stochastic nature of our algorithm, we do not have any theoretical proof that all the collisions are detected. However, in Fig. 14, we present one of the several evaluations which we performed on the collision processing algorithm prior to the integration (see [6] for detailed results). Here, as the object is deformed, we plotted the colliding regions as detected by our method and a naïve $O(n^2)$ method. Results show that our method is able to identify all the *active colliding regions* by evaluating a far fewer number of points. Though this is not same as finding the *exact collisions*, it is good enough for our case, since we can quickly narrow down to these points using temporal coherence. Nevertheless, we miss collisions (see Fig. 13(d)) in some cases of intestine-mesentery due to an entirely different reason. The interpenetrations occurred because we performed only segment-segment collision detection. Note that this technique worked well for intestine-intestine and we don't see any collisions missed here. But the mesentery was modeled as a triangulated mesh and when there are cases of segment-triangle or point-triangle, the above approach in it's present form was not effective. More recently, we extended this approach by adding point-triangle, edge-triangle cases and tested it successfully with a mesentery-like surface.

Qualitative Validation: The prototype simulator was demonstrated to the surgeons at IRCAD on July, 2003. The surgical educators practiced on our simulator and explained to us the good points as well as the drawbacks of the current model. The overall intestine behavior and contact modeling was observed to be very good. Some of the instabilities observed in certain cases of the intestine's motion actually turned out to simulate a patient who is spasmodic (suffering from intestinal convulsions). Although this happens quite frequently in a real patient, this problem can be fixed by increasing the damping of the system. They also suggested that the mesentery should be less elastic. With our mass-spring model, the characteristics demanded by the surgeons can be obtained simply by tuning the simulation parameters. Though these parameters can be varied intuitively, it is difficult to incorporate those parameters obtained from biometric studies. Finally, a small error was detected in

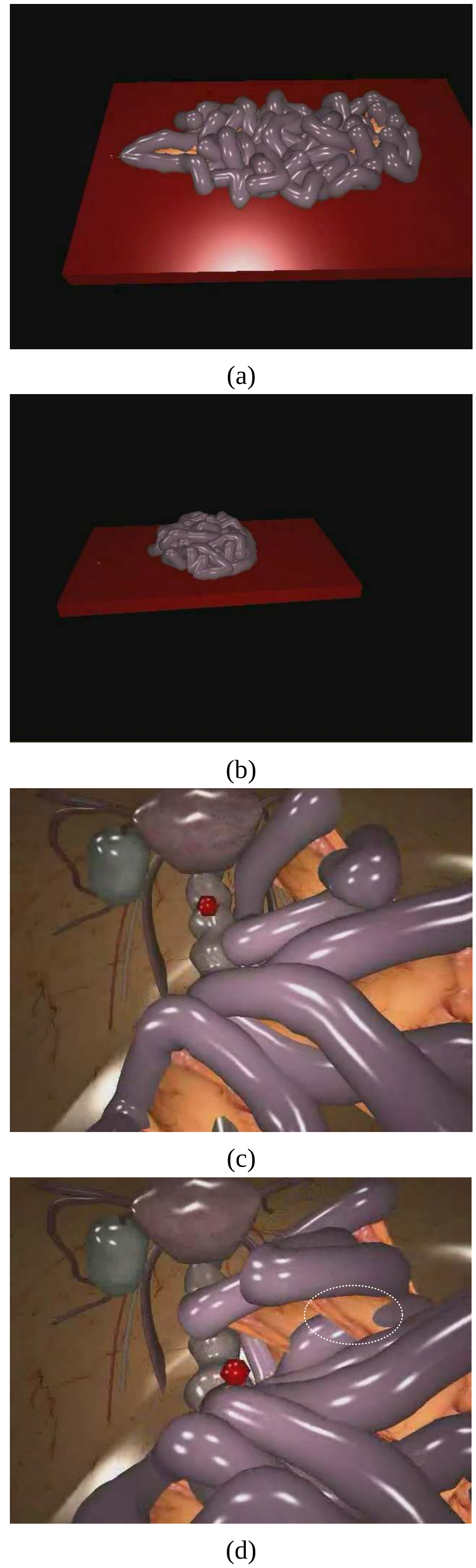


Fig. 13. Snapshots from our intestinal surgery simulator. (a) Intestine and mesentery on a plane pulled by a probe. (b) Stable rest position. (c) Inside the virtual abdominal cavity. (d) Case when collision detection fails (see encircled region).

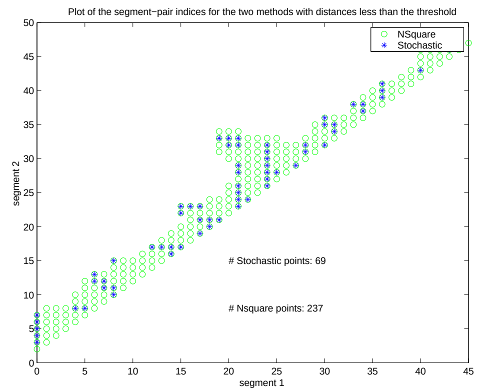


Fig. 14. Quantitative evaluation of the collision detection method. Points in green (light circles) by $O(n^2)$ and blue (dark points) by our approach.

the geometric design, i.e., the mesentery should have a zero width at the two extremities, where the intestine is directly attached to the main vessels.

VIII. Conclusions

Our novel anatomical model greatly simplified the problem of simulating a complex deformable organ in real-time. The collision model is able to efficiently determine the active regions of self-collisions and provides a realistic and stable response. Our rendering technique generates fast and high quality image sequences with no tessellation artifacts. All our three contributions were integrated into a surgical simulator platform with real-time performance. The feedback from the surgeons is positive in general with suggestions for improvements. The corrections are not hard to incorporate in our system and we are currently working on them. We believe that our simulator can be used as a generic trainer that can mimic the behavior of the organs very well thus eventually replacing the practice on animals. The results encouraged us to include triangle tests for more robust collision detection. We are also working on adapting continuous collision detection techniques for handling fast motion of thin objects such as membranes, vessels, etc.

Acknowledgments

This work is supported by INRIA (National Institute for Research in Computer Science and Control) under the ARC SCI grant (Intestine Surgery Simulator project). The authors would like to thank Luc Soler (IRCAD) for his help on anatomy and for facilitating the feedback from the surgeons.

References

[1] A. P. Fine, *Patient Information: Laparoscopic Colon Surgery*. Society of Laparoendoscopic Surgeons, Inc., 2000. [Online]. Available: http://www.sls.org/patientinfo/colon.html

[2] S. Sgambati and G. Ballantyne, “Minimally Invasive Surgery for Diseases of the Colon and Rectum: The Legacy of an Ancient Tradition,” in *Laparoscopic Colectomy*, R. Jager and S. Wexner, Eds. NY: Churchill & Livingstone, 1995, pp. 13–23.

[3] L. Augusten, R. Bowen, and M. Rouge, “Pathophysiology of the Digestive System,” 2002. [Online]. Available: http://arbl.cvmbs.colostate.edu/hbooks/pathphys/digestion

[4] C. for Videoendoscopic Surgery, *Laparoscopic Colectomy*. Dept. of Surgery, University of Washington, 1998. [Online]. Available: http://depts.washington.edu/cves/lapcol.html

[5] P. Meseure, J. Davanne, L. Hilde, J. Lenoir, L. France, F. Triquet, and C. Chaillou, “A Physically based Virtual Environment Dedicated to Surgical Simulation,” in *Surg. Sim. & Soft Tis. Model.* Springer, 2003, pp. 38–46.

[6] L. Raghupathi, V. Cantin, F. Faure, and M.-P. Cani, “Real-time Simulation of Self-collisions for Virtual Intestinal Surgery,” in *Surg. Sim. & Soft Tis. Model.* Springer, 2003, pp. 15–26.

[7] L. Grisoni and D. Marchal, “High Performance Generalized Cylinder Visualization,” in *Proc. Shape Model. Intl. ’03*. IEEE CS Press, 2003, pp. 257–63.

[8] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović, “Interactive Skeleton-Driven Dynamic Deformations,” in *Proc. SIGGRAPH ’02*. ACM Press, 2002, pp. 586–93.

[9] G. Debunne, M. Desbrun, M. P. Cani, and A. H. Barr, “Dynamic Real-time Deformations Using Space and Time Adaptive Sampling,” in *Proc. SIGGRAPH ’01*. ACM Press, 2001, pp. 31–36.

[10] E. Grinspun, P. Krysl, and P. Schr¨oder, “CHARMS: A Simple Framework for Adaptive Simulation,” in *Proc. SIGGRAPH ’02*. ACM Press, 2002, pp. 281–90.

[11] M. Bro-Nielsen, “Finite Element Modeling in Medical VR,” *Journal of the IEEE*, vol. 86, no. 3, pp. 490–503, 1998.

[12] D. L. James and D. K. Pai, “ArtDefo - Accurate Real Time Deformable Objects,” in *Proc. SIGGRAPH ’99*. ACM Press, 1999, pp. 65–72.

[13] G. Székely, C. Brechb¨uhler, R. Hutter, A. Rhomberg, N. Ironmonger, and P. Schmid, “Modelling of Soft Tissue Deformation for Laparoscopic Surgery Simulation,” *Medical Image Analysis*, no. 4, pp. 57–66, 2000.

[14] S. Cotin, H. Delingette, and N. Ayache, “A Hybrid Elastic Model for Real-time Cutting, Deformations, and Force Feedback for Surgery Training and Simulation,” *The Vis. Comp.*, vol. 16, no. 8, pp. 437–52, 2000.

[15] P. Meseure and C. Chaillou, “A Deformable Body Model for Surgical Simulation,” *J. Vis. Comp. Anim.*, vol. 11, no. 4, pp. 197–208, 2000.

[16] M. Kauer, V. Vuskovic, J. Dual, G. Székely, and M. Bajka, “Inverse Finite Element Characterization of Soft Tissues,” in *Proc. MICCAI*. Springer, 2001, pp. 128–136.

[17] J. C. Lombardo, M. P. Cani, and F. Neyret, “Real-time Collision Detection for Virtual Surgery,” in *Proc. Comp. Anim. ’99*. IEEE CS Press, 1999, pp. 82–91.

[18] L. France, A. Angelidis, P. Meseure, M. P. Cani, J. Lenoir, F. Faure, and C. Chaillou, “Implicit Representations of the Human Intestines for Surgery Simulation,” in *Model. & Sim. for Comp. Aid. Med. & Surg.*, ser. Proc. ESAIM vol. 12, M. Thiriet, Ed. SMAI, Paris, 2002, pp. 42–47.

[19] L. France, J. Lenoir, P. Meseure, and C. Chaillou, “Simulation of a Minimally Invasive Surgery of Intestines,” in *Proc. 4th Virtual Reality Intl. Conf.*, Laval, France, June 2002.

[20] G. Bradshaw and C. O’Sullivan, “Sphere-Tree Construction Using Dynamic Medial Axis approximation,” in *Proc. Symp. Comp. Anim.* ACM Press, 2002, pp. 33–40.

[21] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi, “I-COLLIDE: An Interactive and Exact Collision Detection

System for Large-Scale Environments," in *Proc. ACM Symp. Inter. 3D Graphics*. ACM Press, 1995, pp. 189–96.

[22] S. Gottschalk, M. Lin, and D. Manocha, "OBBTree: A Hierarchical Structure for Rapid Interference Detection," in *Proc. SIGGRAPH '96*. ACM Press, 1996, pp. 171–80.

[23] I. J. Palmer and R. L. Grimsdale, "Collision Detection for Animation Using Sphere Trees," *CG Forum*, vol. 14, no. 4, pp. 105–16, 1995.

[24] G. van den Bergen, "Efficient Collision Detection of Complex Deformable Models Using AABB Trees," *J. Graphics Tools*, vol. 2, no. 4, pp. 1–14, 1997.

[25] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan, "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs," *IEEE TVCG*, vol. 4, no. 1, pp. 21–36, 1998.

[26] G. Debunne and S. Guy, "Layered Shells for Fast Collision Detection," 2004, to be published.

[27] M. Lin and J. Canny, "Efficient Collision Detection for Animation," in *Proc. 3rd EG Anim. & Sim. Workshop*, 1992.

[28] G. Agin and T. Binford, "Representation and Description of Curved Objects," *IEEE Trans. Comp.*, vol. 25, no. 4, pp. 439–49, 1976.

[29] J. Bloomenthal, "Calculation of Reference Frames Along a Space Curve," in *Graphics Gems*, A. Glassner, Ed. NY: Academic Press, 1990, pp. 567–71.

[30] "nVIDIA OpenGL Vertex Weighting." [Online]. Available: http://developer.nvidia.com/object/GL_EXT_vertex_demo.html

[31] J. Bloomenthal, "Medial-Based Vertex Deformation," in *Proc. Symp. Comp. Anim.* ACM Press, 2002, pp. 147–51.

[32] C. Blanc and C. Schlick, "Extended Field Functions for Soft Objects," in *Proc. Implicit Surface '95*, 1995, pp. 21–32.

[33] "L'Institut de Recherche contre les Cancers de l'Appareil Digestif, Strasbourg." [Online]. Available: http://www.ircad.org

[34] S. Uno and M. Slater, "The Sensitivity of Presence to Collision Response," in *Proc. IEEE VRAI Symposium*. IEEE CS Press, 1997, pp. 95–103.

[35] D. H. Eberly, *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*. Morgan Kaufmann, 2000.

[36] D. Baraff and A. Witkin, "Large Steps in Cloth Simulation," in *Proc. SIGGRAPH '98*. ACM Press, 1998, pp. 43–54.

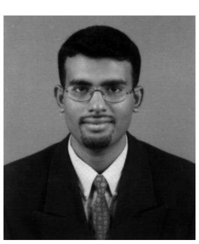
[37] J. Platt and A. Barr, "Constraints Methods for Flexible Models," in *Proc. SIGGRAPH '88*. ACM Press, 1988, pp. 279–88.

[38] T. Giang, G. Bradshaw, and C. O'Sullivan, "Complementarity Based Multiple Point Collision Resolution," in *Proc. Fourth Irish Workshop on Computer Graphics*, 2003, pp. 1–8.

[39] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*. NY: Academic Press, 1990.

[40] L. Piegl and W. Tiller, *The NURBS Book*. NY: Springer, 1995.

[41] G. Elber and E. Cohen, "Second Order Surface Analysis Using Hybrid Symbolic and Numeric Operators," *ACM TOG*, vol. 12, no. 2, pp. 160–78, 1993.



Laks Raghupathi received his BSEE from BITS, Pilani, India (1999) and MSEE from the Univ. of Texas at Arlington, USA (2002). He is currently pursuing his Ph.D. at INP Grenoble, France and is a member of the GRAVIR/IMAG laboratory. His research interests include physically-based modeling, collision detection and various other aspects of real-time computer graphics. He is a member of the Tau Beta Pi and Eta Kappa Nu.

Laurent Grisoni received his diploma in Computer Science from ENSEIRB national engineering school in 1996, and his Ph.D. from the Univ. of Bordeaux I in 1999. He is currently an Assistant Professor at the Polytech'Lille engineering school, Univ. of Lille 1, and is a member of the INRIA ALCOVE project. His research interests include spline and implicit modeling, animation and multiresolution aspects.

François Faure completed his Ph.D. in Computer Science at the Univ. Joseph Fourier, Grenoble, France in 1997, where he is currently an Assistant Professor. He previously studied Mechanical Engineering at the École Normale Supérieure de Cachan. His research interests are about physically-based animation of complex scenes.

Damien Marchal received his Masters from Univ. of Lille 1 in 2003. He is currently working on his Ph.D. at the LIFL lab and is a member of the INRIA ALCOVE project. His research interests include collision detection, deformable models, hardware accelerated computation.

Marie-Paule Cani is a Professor of Computer Science at INP Grenoble, France. A graduate from the École Normale Supérieure, she received her Ph.D. from the Univ. of Paris Sud in 1990. Her main research interests cover physically-based simulation, implicit surfaces applied to interactive modeling and animation and the design of layered models incorporating alternative representations and LODs.

Christophe Chaillou is a Professor at the Polytech'Lille engineering school, Univ. of Lille 1 since 1995 and is the research director of the INRIA ALCOVE project. His research interests include haptic peripheral design and collaborative virtual environments.