



Published in final edited form as:

Int J Med Robot. 2009 September ; 5(3): 341–353. doi:10.1002/rcs.266.

Using the PhysX engine for Physics-based Virtual Surgery with Force Feedback

Anderson Maciel, Ph.D.[postdoctoral research associate],

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil

Tansel Halic, B.Eng.[Ph.D. student],

Department of Mechanical, Aerospace and Nuclear Engineering, Rensselaer Polytechnic Institute, Troy, USA

Zhonghua Lu, B.Eng.[Ph.D. student],

Department of Mechanical, Aerospace and Nuclear Engineering, Rensselaer Polytechnic Institute, Troy, USA

Luciana P. Nedel, Ph.D.[assistant professor], and

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil

Suvranu De, Sc.D.[associate professor]

Department of Mechanical, Aerospace and Nuclear Engineering, Rensselaer Polytechnic Institute, Troy, USA

Abstract

Background—The development of modern surgical simulators is highly challenging as they must support complex simulation environments. The demand for higher realism in such simulators has driven researchers to adopt physics-based models which are computationally very demanding. This poses a major problem since real time interactions must permit graphical updates of 30 Hz and a much higher rate of 1 kHz for force feedback (haptics). Recently several physics engines have been developed which offer multi-physics simulation capabilities including rigid and deformable bodies, cloth and fluids. While such physics engines provide unique opportunities for the development of surgical simulators, their higher latencies, compared to what is necessary for real time graphics and haptics, offer significant barriers to their use in interactive simulation environments.

Methods—In this work, we propose solutions to this problem and demonstrate how a multimodal surgical simulation environment may be developed based on NVIDIA's PhysX physics library. Hence, models that are undergoing relatively low frequency updates in PhysX can exist in an environment that demands much higher frequency updates for haptics. We use a collision handling layer to interface between the physical response provided by PhysX and the haptic rendering device to provide both real time tissue response and force feedback.

Results—Our simulator integrates a bimanual haptic interface for force-feedback and per-pixel shaders for graphics realism in real time. To demonstrate the effectiveness of our approach, we present the simulation of the Laparoscopic Adjustable Gastric Banding (LAGB) procedure as a case study.

Conclusions—To develop complex and realistic surgical trainers with realistic organ geometries and tissue properties demands stable physics-based deformation methods which are not always

compatible with the interaction level required for such trainers. We have shown that combining different modeling strategies for behavior, collision and graphics is possible and desirable. Such multimodal environments enable suitable rates to simulate the major steps of the LAGB procedure.

Keywords

Computer Graphics; Interaction techniques; Physics-based simulation; Minimally Invasive Surgery; User Interfaces—Haptic I/O

1. Introduction

Physics-based interactions are necessary to improve the realism of modern surgical simulators and allow the response to surgical interventions be computed in a physiologically relevant manner. For real time graphical rendering an update rate of at least 30 Hz must be maintained, whereas for stable force feedback (haptics) a much higher update rate of the order of 1 kHz is necessary. The availability of new powerful graphics hardware is insufficient to meet the demands of such physics-based surgical simulators, which must now include interactions of surgical tools with multiple deformable objects, bleeding and smoke generation due to cautery procedures, as well as tissue approximation procedures such as suturing and stapling. Recently several physics engines have been developed which offer multi-physics simulation capabilities including rigid and deformable bodies, cloth and fluids. While such physics engines provide unique opportunities for the development of surgical simulators, their higher latencies, compared to what is necessary for real time graphics and haptics, offer significant barriers to their use in interactive simulation environments.

In this work, we propose solutions to this problem and demonstrate how a multimodal surgical simulation environment may be developed based on NVIDIA's PhysX physics library. PhysX, which can be accelerated using the physics processing unit (PPU) or CUDA-enabled GeForce graphics processing unit (GPU) – provides an optimized set of methods for physics-based simulation. Our simulator integrates a bimanual haptic interface for force-feedback and shaders for graphic realism in real time. A Model-View-Controller architecture, taking advantage of the parallel computation of modern GPU and multi-core CPU, has been developed to integrate PhysX to exploit the hardware to the maximum. In this integration, heterogeneous models – triangle mesh, tetrahedron mesh, rigid, deformable, and articulated – cohabit in the environment and allow equally heterogeneous 3D interaction modes: line- and point-based collision detection, with or without force-feedback, touching, picking and manipulation. In addition to the development of the heterogeneous simulation environment, another contribution of this paper is to demonstrate how lower frequency threads, as PhysX implicit integration, can fit in a much higher frequency interactive context, such as haptic rendering. Such unlimited combination of models, modes and techniques, though not trivial, make it possible to use the best solution available for each particular problem, opening up endless possibilities of what can be achieved with current hardware technology.

To demonstrate the effectiveness of our approach, we present the simulation of the Laparoscopic Adjustable Gastric Banding (LAGB) as a case study. LAGB is a complex minimally invasive surgical procedure indicated for patients with morbid obesity. It consists of placing a flexible band around the stomach to constrict the food passage and produce an early satiety sensation. The motivation for this specific surgical procedure is the high cost in training surgeons and the increasing demand for it.

In this article we present an overview of the PhysX library, introduce surgery simulation with haptics and present our simulator. We then introduce a case study – showing how we developed and modeled anatomy and physiology for the simulator, followed by a discussion of the results and conclusions.

1.1 Related work

Real-time physics-based surgical simulation is computationally very demanding [1]. For a review of current literature on surgical simulation, please refer to [2]. In summary, geometry-based models developed by e.g., Basdogan et al. [3] and [4], tend to sacrifice the physical realism of the deformation process to achieve real time performance. Physics-based techniques that take into account the underlying mechanics of soft tissue deformation using mass spring networks [5] or finite elements [6,7] have been developed. Techniques such as condensation [8] and precomputation [9] have been used to accelerate finite element computations. Meshfree methods such as the point associated finite field (PAFF) offer yet another alternative to finite element techniques and several methods have been presented to reduce computational costs [1]. Hardware acceleration has also been used to achieve real time rates [10].

As the capability of computing resources has increased over the years, so have the complexities of surgical scenes being simulated. A typical surgical simulation scene now involves much more than just tool-tissue interaction. Simulations of multiple organs, medical devices, sutures, coupled fluid flows due to bleeding and modeling of physiological consequences of surgical procedures are now becoming common. To achieve a certain degree of realism in such complex surgical interactions, the simulation environments need to support the following: (1) heterogeneous scenes composed of different states of matter (solids, liquids, and gases); (2) complex geometry and material properties of objects within the scene; (3) dynamic and real-time interaction (palpation, cutting, etc.) between virtual objects and tools physically manipulated by the user; and (4) multimodal (visual and haptic) rendering of the results to the user. It is clear that it is not possible to satisfy all the four requirements using a single modeling technique. It is therefore necessary to use heterogeneous techniques, each optimized for one or more of the tasks. Real-time physics engines, originally developed for game development, offer a unique platform to develop such heterogeneous simulation scenarios.

In the literature, surgical simulators using physics engines are not very common. This happens mainly because the existing physical engines are often focused on a reduced set of physical laws, which is not enough for a full surgery simulation. Another reason is that they are implemented aiming either at producing fast less accurate animations or very accurate but computationally demanding behaviors which cannot be produced in real time. Yet, a few examples of physics engines exist which are used for surgery simulation [11]. However, these are not general physics engines for games. They are either custom developments in which the developers have the full control of the physical laws they are implementing or general engines allowing plugging in new physical laws using software containers. The former are not consolidated engines and demand great investment in development, while the latter are too much general and cannot be optimized in terms of hardware use.

Physics engines that are generally targeted for game development are not specifically concerned with the intricacies involved in surgical simulation. Still, the problems that need to be tackled are common. For instance, the challenges involved in interaction of an avatar and its cloth in a game scene is very similar to a surgical scene where the surgeon interacts with a membrane. Such similarities are explored in FIGURE 1.

Recently, several physics engines have been developed. Some of these are available commercially, have closed source or are under limited license such as Havok [12], PhysX [13] and Newton Game Dynamics. Some other engines are open source and available for free such as ODE, Open tissue, Dynamechs, True Axis, Tokamak, Bullet, CMS-Labs Vortex, JIGLIB and BOX2D [14]. For more information about the simulator capabilities, we encourage the reader to look into the work of Seugling and Rölin [15] and Boeing et al. [16]. They compared the available physics engines and described the features of each engine. One important conclusion is that only a few of them have the capability to simulate the surgical scenario.

In this regard, Marks et al. [17] investigated the game engines and looked into the adaptability of these engines to surgical simulation. They chose three game engines presenting functionalities that allow for the development of surgical simulations. Id Tech4 [18], Unreal Engine [19] and Source Engine [20] were chosen for further evaluation. These engines were evaluated in terms of editing, content and gameplay. More specifically for the surgical case, their assessment focuses on the maintainability of the surgical task. They analyze whether the engine supports change in surgical procedures, creation of surgical scenarios, simplified incorporation of other custom models, multiple users in simultaneous interactions and, finally, analyze the quality of physics-based interaction.

Physics-based interaction is at the intersection of physics-based engines and real-time haptic interaction. However, a good compromise between them is not easily established, as both require massive computation at different frequencies. In previous works, we dealt with the problem of rendering haptic feedback when very complex triangle meshes are involved [21,22]. The problem is often solved at the level of collision detection using optimized algorithms. Collision detection involves checking the geometry of the target objects for interpenetration, we refer to [23] and [24] for detailed surveys on collision detection.

2. Computational Methods

In section 2.1 we discuss techniques of employing the PhysX library to simulate various entities- soft bodies, rigid bodies and fluids- that are of interest in surgical simulation. In section 2.2 we point out specific issues related to multimodal surgical simulation involving haptics. One of the most critical problems is to be able to maintain various update rates for collision detection, physics-based interaction, visual and haptic rendering on the same model data structure. We present an extended version of a model-view-control (MVC) framework to resolve this issue. In section 2.3 we present concrete examples of how anatomy and surgical instruments may be modeled using various PhysX objects. Finally, in section 2.4 we present our experiences in developing a procedural simulator for laparoscopic adjustable gastric banding (LAGB).

2.1. Using PhysX for interactive simulations

PhysX is a physics-based framework originally developed by Ageia, which was acquired in early 2008 by NVIDIA. Subsequently, NVIDIA added their GPU support in the PhysX framework to harness the capability of their general purpose computation framework CUDA [25]. In general, the PhysX framework supports real-time simulation of various physical entities such as rigid bodies, soft bodies, cloth, joints, fluids, springs, and characters, as well as collision detection and response algorithms with a focus on application to the gaming industry. At the present time, the framework abstracts the hardware use from developers seamlessly and uses GPU or GPUs (if the system has multiple GPU cores) for the simulation without coding.

PhysX [13] is a simulation engine applying position based dynamics [26]. The framework provides functionalities revolving around the provided rudimentary physical objects such as rigid bodies and joints to more complicated physical objects such as soft bodies, fluids and cloth (see FIGURE 2). PhysX mainly differs from game engines in the sense that it does not provide high fidelity rendering and visualization. All rendering and visualization, and tasks related to network communication, user interactivity, audio, and so on, is left to the developer.

In PhysX, entities are created as scene objects (*NxScene*). Scene objects are initialized by the *NxPhysicsSDK*. There can be more than one scene object at a time and each scene object is responsible for simulating the behavior of the constituent objects.

The basic PhysX object is called an *actor*. There are two different actors in PhysX: static and dynamic. As the names imply, static actors are attached to a particular location and stay stationary most of the time in the scene. Conversely, dynamic actors are influenced by the forces in the environment and also influence other objects in the environment when collision happens. Collisions take place between the actors whose shapes may vary from simple primitives such as box, capsules, spheres, etc. to more complex geometries such as convex mesh or triangle mesh.

The center of mass of each actor is computed when the actor is created. Furthermore, the actor is moved and rotated based on the simulation dynamics referred to this center. In the framework, actors have a body only if they are dynamic entities. The body is associated with *NxBodyDesc* class which gives details regarding the body specification such as mass, linear and angular velocity, and local pose with respect to the center of mass. The information also includes damping along the linear movement and angular movement. Actor body description also contains an inertia tensor that is calculated from its shape and density by the framework or it can be explicitly set by computing the tensor manually. In the simulation, external forces, such as direct forces and impulse forces, can be applied to an actor body and are smoothed, so that the resultant force response effect can be controlled. Generally, in games, actors are primarily designed for rigid body elements.

PhysX provides *NxJoint* class to simulate joint structures. With joints, actors can be connected and motions can be limited within a desired range. There are several predefined joints available (Spherical, Revolute, Prismatic). Apart from the existing joints, PhysX allows to define custom joints with the *NxD6Joint* class. This encompasses all existing joints and provides more functionality to create and freeze the degrees of freedom or limit the range of motion of each joint axis. The framework also provides driver mechanism to simulate external joint movement.

PhysX fluids are essentially based on the particles approach. In this technique, the fluid is considered as particles that have density, velocity, position and life time states. The fluid is characterized by properties such as stiffness, damping, viscosity, etc. The particles interact with each other based on two different approaches, depending on user choice: a smoothed particle hydrodynamics (SPH) approach and a simple mode, where inter particle forces are not accounted for. Based on the desired fluid physics, *NxFluidDesc* identifies particle interaction. Currently, PhysX fluids can interact with PhysX cloth, soft bodies, and rigid bodies. However, fluid-fluid interaction is not taken into consideration.

PhysX cloths consist of particles that are constrained within the edges of a shape using bending and stretching parameters. *NxClothDesc* offers additional parameters such as damping, density, friction, pressure for closed cloths to simulate rather stiff behavior (e.g., rug) or more elastic behavior (e.g., skirt). In PhysX cloth, the collision detection takes place between a body and the cloth vertices. The collision detection sensitivity and the response to

collisions can be manipulated by the thickness of the cloth and collision response coefficients, respectively. One of the remarkable characteristics of cloth is that it may be torn which can be controlled with either user supplied parameters that define tearing factor or with explicit function calls.

Soft body simulation is one of the most notable features of the PhysX framework in the context of this work. The framework simulates any given topological elastic body. In general, it differentiates the simulation mesh geometry from the mesh which is used for rendering the soft body. Therefore, the SDK uses a tetrahedral mesh for the internal structure, which is encapsulated by the surface mesh of the body. The method of integration used is semi-implicit. Verlet integration is applied to compute the new positions of each tetrahedron node. The vertices' new positions under external and internal forces (from springs connected to the tetrahedrons vertices) are handled explicitly. In addition to this, the constraints solver is employed in the simulation loop where the solver iterates over constraints by a pre-defined number of iterations which is explicitly set in the soft body description. This type of scheme where both explicit and implicit integration methods are used at the same time is called semi-implicit scheme [26]. In the constraint solver part, the non-linear Gauss-Seidel type iteration is used where each constraint is handled individually. The solver actually resolves the collisions as well as pre-determined constraints after the vertices are projected to their next time step positions by Verlet integration. Constraints solver both copes with fixed constraints and collision constraints. For instance, the stretching stiffness which defines a stretching constraint normalized between zero and one along the edges of a tetrahedron, or volume constraints which create a dependency between the initial reference volume and the current volume can be regarded as fixed constraints. Thus, as stiffness and the number of iterations are increased, both volume and stretching results in stiffer behavior and a slower frame rate. When the collision exists in a given step, the new vertex positions are updated with respect to the generated collision constraints. At the end of the simulation loop, the new positions comply with the fixed and collision constraints, which are then updated.

There are several parameters that influence physics of the soft body significantly. Volume and stretching stiffness of the tetrahedrons specify the internal structure. On the other hand, damping and friction have more impact on the global response of the soft body. Naturally, the equation solver and the number of iterations have substantial effect on the softness and hardness of the body. Moreover, the SDK allows attaching rigid internal shapes to the bodies (a kind of skeleton) that can strengthen the body and provides more realism when the body has a skeleton in reality.

2.2. Surgery simulation with haptics

Force feedback [27] is an essential component of interactive simulations such as minimally-invasive surgery simulation [3] where the user interacts with soft deformable organs using long slender surgical tools. The simplest paradigm of haptic interaction is the use of a point-based representation of the haptic cursor. However, this is unrealistic for many applications including laparoscopic surgery simulation, where it may be necessary to retract an organ or appendage while grasping another one or to slice through tissue, fat or fascia using blunt or cautery instruments. For such kind of applications, it is natural to represent the haptic cursor as a line or geometry. However, for deformable organ models, such ray- and geometry-based representations are prohibitively expensive if realistic organ models are used, and realistic interaction is necessary.

There are many existing interaction approaches that are aimed at reaching visually interactive speeds, i.e., around 30 frames per second, for real time applications. However, in virtual environments involving haptics, a much higher update rate of at least a few hundred

frames per second is necessary to render stable force feedback. If physics-based computations are to be performed as part of the model behavior, then this places severe demands on the efficiency of the algorithms.

We have developed a framework based on the Model-View-Controller (MVC) pattern [28] which allows for the computation, visualization and interaction with physics-based models. MVC is a design and architectural pattern used in software engineering. It aims at isolating data, business logic and how information is displayed. As a result, changes in the visual appearance of the application are independent of the business rules and vice-versa. In MVC, the *Model* represents the application data; the *View* corresponds to elements of the user interface such as text, widgets, 3D rendering areas and so on; and the *Controller* manages the data manipulation and modification applying the business rules. The controller reads data from the model, changes them, and writes them back to the model so that they are available to be viewed or modified by other controllers. Many controllers and/or viewers are allowed simultaneously, while all data is seen as a unique module. In our framework, *Data* holds geometric and material information, *Viewers* display *Model* information on graphics windows and user interface elements as key-pressing or mouse clicking, and *Controllers* modify the *Model* – e.g., the collision detection controller inspects the geometry and flags the penetrating triangles, the physics controller applies physical rules to modify the forces, accelerations, velocities and positions in the *Model*, and so on. While similar control schemes have been used before [26], they usually prioritize the physics simulation over other control tasks keeping it apart from the MVC structure. The uniqueness of our approach is that physics is also encapsulated in a controller, managed by PhysX, resulting in a highly efficient use of the hardware resources.

The haptics controller allows communication of the model with the force-feedback device (the Phantom). This controller is not responsible for calculating the forces. The interaction forces are calculated by the physics and collision detection controllers and stored in the model. Hence, the haptics controller becomes a very fast scanner which reads forces and positions from the model and makes the matrix conversions necessary before calling back the device driver. Such conversions are often necessary to register 3D spaces and to apply scaling between worlds. Since we have used the Sensable Phantom Omni™ as force-feedback device, the OpenHaptics library is the interface used by this module to communicate with the device. Such asynchronous design allows the force feedback to be updated at a frequency of more than 10 kHz . Notice that it is a consensus in the haptics literature that at least 1 kHz is necessary to ensure smooth transitions. Our framework is designed in such a way that multiple execution threads run in parallel and concurrently at different rates and in different cores on the latest CPUs. This permits, for example, that haptic computation is transparently made at over 1000 Hz while a custom explicit integrated mass-spring system runs at lower frequencies, below 400 Hz . At the same time, graphics rendering is made at 30 Hz . This is possible because with MVC only the controllers modify the model (see FIGURE 3). While this approach allows simulating some surgical steps, it cannot completely simulate a surgery, as due to computational complexity only a few entities can be physically based and increasing the number of entities degrades the performance dramatically. To keep the performance at the high level required for the task we replaced the custom explicitly integrated mass-spring model by PhysX, which is implicitly integrated and thus much more stable. The trade-off is that implicit integration methods are more computationally demanding. With our current soft bodies, the physics thread runs at below 20 Hz . At the same time, collision detection, which is optimal at a much higher frequency depending on the dynamics of the simulation (objects moving fast may require over 1000 Hz while calm scenes would accept collisions to run at the same pace as physics), and realistic graphical rendering, which runs at a constant frequency of 30 Hz ,

must be maintained. For this to happen, the model data structure, viewer and interaction controllers must remain unchanged, which is not straightforward.

We solve this critical problem by developing an extended version of the MVC framework as presented in FIGURE 4. In this framework, we include PhysX as a new controller that modifies the model geometry already present in the data structure. Every PhysX step is performed on a separate thread at its optimal frequency, determined by the physical properties of the materials involved. At the end of each step, the model geometry is updated at once, similarly to a swap buffer step.

2.3. Modeling anatomy, physiology and tools

For the assessment of our simulator we consider a realistic scenario of a laparoscopic surgical procedure. One of the goals of this experiment is to evaluate our algorithms in the practical situation of a virtual laparoscopic intervention and to explore the responsiveness of the system. An additional aim is to demonstrate that high frequency haptic response and rigid and deformable objects simulated at a lower frequency by implicit integration can co-exist in a fully interactive real time haptic environment.

The scenario consists of a partial model of the interior of the abdominal cavity where a few organs are visible as they would be visible through the laparoscope. The organs include a deformable liver, a deformable stomach and a rigid spleen, with the peritoneum in the background.

The organ models are obtained from the segmented data of the VIP-Man, which is a series of high resolution photographic images from the Visible Human Project dataset [29]. Images we used are 1878 slices of 1760×1024 pixels in raw format. Pixel size is 8 bits with a value range from 0 to 71 according to the labels in TABLE 1. In the original images, distance between slices is 3 mm, and between pixels it is 0.35 mm, however we down sampled them to pixels of 1.4 mm for performance issues. See a slice example in FIGURE 5. As the stomach is labeled in three different parts – wall, content and mucosa – we re-labeled all three to the same value. The esophagus has also been blended with the stomach to avoid non-realistic discontinuities. Then we used the VTK [30] implementation of the marching cubes algorithm [31] to extract the organs surfaces. As the marching cubes algorithm is based on an isosurface, we select the tissue to be reconstructed setting image values in the range of the targeted tissues to 1024 while all others are set to 0. An isosurface is computed with the value 512, marching cubes places triangles at the cube-surface intersections and the resulting dataset is filtered to reduce number of triangles before being written to a file in standard formats like obj and stl. The liver, stomach and spleen are meshes composed respectively of 2484, 1863 and 1040 triangles. The liver and stomach are simulated by PhysX soft bodies with around 3k tetrahedrons.

The peritoneum is the internal wall of the abdominal cavity and is not labeled in the VIP-Man. Hence a different method was used to create the mesh. We drew outlines (FIGURE 6) on selected slices of the cross-sectional Visible Human CT dataset, composed of a series of points which are dense in curved areas and sparse in areas of lesser curvature. We started with the most distal cross-sectional layer, manually outlined the shape on the image, then copied the outline and dragged the copy along the proximal direction to the next slice. Control points were edited (moved\added\united) on the new slice to outline the peritoneum shape according to the corresponding anatomy. Triangles were created to mesh the space in-between the two outline curves. The process was repeated for all subsequent layers until the peritoneum was no longer visible for the current layer. FIGURE 7 shows a schematic of the process.

The resulting peritoneum tightly encloses the organs. However, for surgery, the abdomen is insufflated with CO_2 to provide room for the laparoscope and other tools. To simulate the shape when the peritoneum is insufflated, we edited the control points to enlarge the front peritoneum. The resultant peritoneum, as shown in FIGURE 8a, contains 213 vertices and 384 triangles. We further use texture mapping and per pixel illumination models to render realistic graphics for all organ models, as shown in FIGURE 8b. As we will see in the next section, the surgical scenario we simulate contains a band, known as the LapBand[®] developed by Allergan, Inc. (see FIGURE 9a) that is placed around the stomach model. Modeling of this band is not trivial. The LapBand[®] has two distinct parts having different physical behavior. The thick and stiffer part, which is identified as the upper part, has an adjustable ring. The long part, which looks like a tube, is more flexible, threads through the stiffer part, and connects the band to the port in the actual surgery. The thick part is almost inextensible and tends to return to its original curved shape when released. The long part is more elastic and highly deformable. Since the two regions have different material and structural properties, we split the band model into two different meshes so that it presents plausible physical behavior. It is worthwhile to mention here that a limitation of PhysX is that soft bodies must be isotropic and homogenous, i.e., it does not allow the definition of two different material properties within the same soft body. Having two distinct meshes gives us the flexibility to define physical parameters independently between them. For instance stretching stiffness is set to the maximum value for the thick part. In addition, the thick part has a high factor of volume conservation, is highly damped, and the global damping (NX_SBF_COMDAMPING) is also highly effective in its deformation. On the other hand, the long part is less stiff and no global damping is used (center of mass damping).

Since the two meshes are detached, some sort of constraint has to be established between them to avoid discontinuous motion when either of the parts is moved or deformed. The solution we propose using PhysX is to attach a rigid body actor to either parts of the band, and then connect them with a distance joint. The distance joint provides constrained distance in each iteration and allows for the motion to be guided through any one of the two meshes. Solver number of iterations per second for the band is set to a higher value which affects the stability of the overall simulation.

The haptic cursor on the screen may take the shape of any surgical instrument. When an instrument is selected it is associated with the Phantom cursor and moves following motion of the handle manipulated by the user. The user can switch between instruments at any time using a menu. Currently we modeled the set of instruments required for the LAGB procedure. This includes the cautery hook, the grasper and the scissors. The cautery hook is modeled as a unique rigid body on PhysX. The grasper and scissors are modeled each as a set of 3 PhysX convex rigid actors. Both have one root part, which moves following the Phantom stylus, and two leaf parts, which perform an open/close motion to cut, in the case of the scissor, and to grasp, in the case of the grasper. All three parts are rigid bodies in PhysX and are connected by a rotational joint. FIGURE 10 illustrates the instrument models.

2.4. Laparoscopic Adjustable Gastric Banding (LAGB) procedural simulation

We use the procedural simulation of the Laparoscopic Adjustable Gastric Banding (LAGB) as a case study. LAGB is a complex minimally invasive surgical procedure which consists of placing the Lap Band[®] around the stomach of a morbidly obese patient to constrict the passage of food which results in early satiation. We simulate the ‘pars flaccida’ technique for the lap-band placement, which involves basically three phases:

Phase 1 Port placement—In this phase five access ports are placed on the abdomen. These are small perforations in the abdominal walls through which the trocars and laparoscope would be introduced. This is followed by pumping carbon dioxide gas into the peritoneal cavity (pneumoperitoneum) which separates the organs and allows better accessibility.

Phase 2 Dissection—In this phase a passage is created behind the upper part of the stomach to pass the band. To perform this step, the liver is retracted using a Nathanson hook liver retractor, the gastro-hepatic ligament (*pars flaccida*) is divided avoiding the hepatic artery followed by the retraction of the gastric fundus and omental fat. The angle of His is subsequently mobilized to create a small window in the phrenoesophageal ligament. The right crus is identified and the peritoneum overlying its lower portion near junction of the left crus is divided.

Phase 3 Band placement—In the next phase a grasper is inserted from the right passing through the opening behind the upper part of the stomach until it can be seen by the angle of His at the left. The band is then inserted into the abdomen and pulled around posteriorly, from left to right, with the passing device or a grasper. The band is then secured and locked in position.

As the first phase is extracorporeal, we focus here on the next two phases. The major steps in this modeling using PhysX are explained below, with FIGURE 11 illustrating these steps:

Modeling phase 2—We modeled a hook cautery (FIGURE 10a) as a rigid body and a blunt dissector or grasper (FIGURE 10b) as a rigid articulated body. The positions and orientations of the surgical instruments (hook cautery, blunt dissector or grasper) are controlled using a Phantom Omni as 3D input device with 6 DOF for each instrument. When the virtual instruments contact a virtual organ, the haptic interface device produces force feedback to the user's hand. The electrosurgical procedure is simulated modeling the temperature increase caused by approaching the cautery to the tissue, removing triangles when vaporization temperature is reached [32]. FIGURE 12 shows the color mapping of the temperature (a), the result of cutting a membranous tissue (b) and the result of cautery applied on a massive organ (c). The dissector can grab parts of the organs to push and pull soft regions in order to remove obstacles from the field of view allowing visual exploration of hidden areas such as the right crus and the angle of His. The virtual laparoscope is controlled with the mouse.

To accomplish these tasks, we combined custom methods and PhysX capabilities. The organ deformation is calculated by PhysX, while the graphics rendering is accomplished using our custom shaders. As soft body to soft body collision detection is not efficient in PhysX, the method described in reference [33] is used for organ to organ collision detection. However, PhysX rigid body to soft body collision detection is very efficient and is adapted to detect contacts between instruments and organs.

Modeling phase 3—The instruments used in phase 2 are also used in this phase. Two graspers are used to manipulate the band. As discussed in the previous section, the band is modeled as two meshes which are coupled and can be manipulated (picked and dragged) with the instruments. Consequently, collision detections between the band and the organs (soft bodies) require a custom method [33] but collisions with the instruments are well dealt with by PhysX. As for picking, when the grasper closes, all colliding triangles are flagged as *picked* by that instrument. Then, as the closed grasper moves around, all picked triangles are kinematically controlled in the PhysX model, i.e., their positions are updated following the displacements of the handle at the beginning of every simulation frame. Finally, the

positions of all nodes adjacent to flagged triangles are recalculated by PhysX to smoothly follow the displacements. Flags are reset when the grasper opens to release the picked triangles.

3. Results

We have implemented an interactive PC-based surgical simulation framework and tested it on an Intel(R) Core™ 2 Quad 2.66GHz machine with a GeForce 8800 GTX graphics card. Customized vertex and pixel programs are used for textured shading, which provides state of the art interactive graphics realism. This simulator utilizes two force feedback devices to provide bimanual interactivity. The hardware setup is shown in FIGURE 13.

The framework relies on a combination of the PhysX physics engine and customized parallel algorithms to render physics, graphics and haptics in real time. It exploits the parallel capabilities of current multi-core CPUs and multiprocessor GPUs to maximize efficiency. Although using game oriented engines has the potential to improve efficiency and physics realism, the greatest challenge was to control the various threads running at different frequencies in the same environment. We have shown that this is possible by extending the MVC pattern in the sense that the many *Controllers* exchange information through the *Model* at the frequency of the slower thread for each pair of threads.

The system displays graphics at 60 Hz, which, for stereo visualization is 30 Hz for each monitor/eye. Such frequencies provide smooth graphical displays with no flickering. The haptic response is provided at frequencies over 1000 Hz, which allows for a vibration-free haptics rendering. The force update in the model, in turn, is bounded by the collision detection and response algorithms, which run at frequencies between 100 to 4000 Hz depending on the number of elements involved: rigid to rigid being the fastest; soft to rigid being slower; and soft to soft being the slowest.

As for the physics processing, while explicit integration methods are simple but require more steps (usually 300 to 600 steps per second are explicitly set for real time with our models) for convergence, PhysX uses implicit methods that are more time consuming *per frame* but are unconditionally stable. Typically, our simulation processes physics at 10 to 20 Hz. It is important to notice that at times between the 10 to 20 Hz updates the tools can still be moving fast (updated at several hundreds of hertz) and then, even if the soft body vertex positions are not updated in that interval, the force feedback resulting from collisions will change at collision frequency as the tools move. Regarding physics implementations on GPU and CPU, in preliminary tests we observed that some models run over ten times faster when the parallel hardware (GPU) is enabled, especially when fluid simulations are involved. However, other models run at the same rate with both GPU and CPU implementations. This happens because the current release of the PhysX driver enables only a small set of its functions to run on the GPU through the CUDA technology. Consequently, to make any conclusion here in this regard is premature. We plan to perform thorough comparative tests with different scenarios as soon as a new driver is released, which is expected still this year.

Another issue is deformation accuracy. To demonstrate that PhysX soft bodies are accurate enough for surgery simulation we simulated the hemisphere of FIGURE 14a. We modeled the same hemisphere with both PhysX and finite elements (FEM). For the PhysX model we used 1258 vertices, 3901 tetrahedra and 4224 links, and for the FEM discretization we used 1715 nodal points. Then we indented the two hemispheres at their poles to simulate the interaction with a surgical tool tip and analyzed the resulting deformation profiles. Results in

FIGURE 14b show that the profile obtained with PhysX is close enough to the one obtained with FEM for the same indentation of the pole.

We further analyze how execution time and stability change with the number of polygons and PhysX parameters as stiffness, density, damping, etc. but especially, how scale changes the physical behavior. We noticed that scaling the original model affects the responsiveness of PhysX due to the higher stability of massive elements in the process of numerical integration of physical behavior. Stiffer models, as expected, require additional steps in numerical integration and add delay to the physics response. Similar behavior occurs when the density of the elements is too low.

Damping the system increases stability, but also increases delays. Due to this two way relation between density, damping and stiffness, as shown in FIGURE 15, if changes in the scale of the model geometry are distributed in the same proportions to the mechanical parameters, then it will cause behavioral disturbances. The scale factors for each property must be defined taking adequate proportions into account. Further details on scalability are discussed in [34].

4. Discussion

The complete LAGB simulation is complex and under development. However, important advances have been made in developing the overall simulation framework based on PhysX. There are many advantages of using a physics engine such as PhysX as outlined in the paper. The results are encouraging and translate easily to other surgical simulation scenarios with complex interacting soft bodies. However, it is important to realize some of the limitations of PhysX, such as the inability to describe multiple material properties of the same object and the lack of a mechanism to modify accelerations and velocities during a simulation step. The PhysX source code is closed and the API does not allow integration of custom algorithms. Finally, the collision detection algorithms do not cover all primitive to primitive tests.

It is important to realize that surgical simulators are not glorified video games. Soft tissue mechanical characteristics must be obtained from actual experimental data. There is significant work being conducted in our lab in this direction [35]. Another important aspect is validation. The simulators must be tested by surgeons and surgical residents to ensure that they indeed result in positive training transfer. In fact, to justify their added costs, they must be more effective than traditional mechanical training tool boxes or video trainers. Such validation efforts are also under way at the Harvard Medical School [36].

Acknowledgments

Grant

Partial support of this work through the R01 EB005807-01 grant from NIH/NIBIB and 151820/2007-4 grant from CNPq-Brazil are gratefully acknowledged.

References

1. De S, Lim Y-J, Muniyandi M, Srinivasan MA. Physically Realistic Virtual Surgery Using the Point-Associated Finite Field (PAFF) Approach. *Presence*. 2006; 15:294–308.
2. Lim Y-J, Jin W, De S. On Some Recent Advances in Multimodal Surgery Simulation: A Hybrid Approach to Surgical Cutting and the Use of Video Images for Enhanced Realism. *Presence*. 2007; 16:563–583.

3. Basdogan, C.; Ho, C-H.; Srinivasan, MA.; Small, SD.; Dawson, SL. Force interactions in laparoscopic simulations: haptic rendering of soft tissues; Medicine Meets Virtual Reality Conference 6; 1998.
4. Delp SL, Loan JP, Basdogan C, Rosen JM. Surgical simulation: an emerging technology for emergency medical training. Presence: Teleoperators and Virtual Environments. 1997; 6:147–159.
5. Kühnapfel UG, Neisius B, Krum HG, Kuhn C, Hübner M. CAD-based Simulation and Modelling for Endoscopic Surgery. Proc. MedTech. 1994
6. Bro-Nielsen M. Finite element modeling in surgery simulation. Proceedings of the IEEE. 1998; 86:490–503.
7. De, S.; Srinivasan, MA. Medicine Meets Virtual Reality MMVR7. IOS Press; 1999. Thin walled models for haptic and graphical rendering of soft tissues in surgical simulations; p. 94-99.
8. Bro-Nielsen M, Cotin S. Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation. Computer Graphics Forum. 1996; 15:57–66.
9. Picinbono G, Delingette H, Ayache N. Non-linear anisotropic elasticity for real-time surgery simulation. Graphical Models. 2003; 65:305–321.
10. Liu, Y.; De, S. CUDA-based real time surgery simulation. In: Westwood, JD.; Haluck, RS.; Hoffman, HM.; Mogel, GT.; Phillips, R.; Robb, RA.; Vosburgh, KG., editors. MMVR16: Medicine Meets Virtual Reality 16. Vol. 132. IOS Press; 2008. p. 275-280.
11. Nourian S, Xiaojun Shen, Georganas ND. Role of extensible physics engine in surgery simulations. IEEE International Workshop on Haptic Audio Visual Environments and their Applications. 2005 Oct.:6. 2005.
12. Havok. Havok Home Page. 2008 online.
13. NVIDIA. PhysX library. 2008 online.
14. Box2D.org. Box2D Physics Engine. 2008 online.
15. Seugling, A.; Rölin, M. Evaluation of Physics Engines and Implementation of a Physics Module in a 3d-Authoring Tool. Umea University; 2006.
16. Boeing, A.; Bräunl, T. Evaluation of real-time physics simulation systems. GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia; ACM, New York, NY, USA. 2007. p. 281-288.
17. Marks, S.; Windsor, J.; Wünsche, B. Evaluation of game engines for simulated surgical training. GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia; ACM, New York, NY, USA. 2007. p. 273-280.
18. Wikipedia. id Tech 4 (Doom 3 Engine). 2008 online.
19. Unreal. The Unreal Engine. 2008 online.
20. Valve. The Source engine. 2008 online.
21. Maciel A, De S. An efficient dynamic point algorithm for line-based collision detection in real time virtual environments involving haptics. Computer Animation and Virtual Worlds (19:2). 2008:151–163.
22. Maciel A, Liu Y, Ahn W, Singh TP, Dunnican W, De S. Development of the VBLaST™: a virtual basic laparoscopic skill trainer. The International Journal of Medical Robotics and Computer Assisted Surgery (4:2). 2008:131–138.
23. Lin, MC.; Gottschalk, S. Collision detection between geometric models: A survey; Proc. of IMA Conference on Mathematics of Surfaces; 1998. p. 37-56.
24. Teschner M, Kimmerle S, Heidelberger B, Zachmann G, Raghupathi L, Fuhrmann A, p. Cani M, Faure F, thalmann NM, Strasser W, Volino P. State of The Art Report Collision Detection for Deformable Objects. 2005
25. NVIDIA. Compute Unified Device Architecture (CUDA) Framework. 2008 online.
26. Müller M, Heidelberger B, Hennix M, Ratcliff J. Position based dynamics. J. Vis. Comun. Image Represent. (18:2). 2007:109–118.
27. Maass, H.; Chantier, BBA.; Çakmak, HK.; Kühnapfel, UG. How to add force feedback to a surgery simulator. In: Ayache, N.; Delingette, H., editors. Proceedings of the International Symposium on Surgery Simulation and Soft Tissue Modeling. 2003 June 12–13. p. 165-174.

28. Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional. 1995
29. Spitzer V, Ackerman MJ, Scherzinger AL, Whitlock D. The visible human male: a technical report. J Am Med Inform Assoc. 1996; 3:118–130. [PubMed: 8653448]
30. Schroeder, W.; Martin, K.; Lorensen, B. The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics. Prentice Hall: 1997.
31. Lorensen WE, Cline HE. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics (Proceedings of SIGGRAPH 87). 1987:163–169.
32. Maciel, A.; De, S. Physics-based Real Time Laparoscopic Electrosurgery Simulation . In: Westwood, JD., et al., editors. Studies in Health Technology and Informatics: Medicine Meets Virtual Reality 16 -parallel, combinatorial, convergent: NextMed by Design. Vol. 132. Amsterdam: IOS Press; 2008. p. 272-274.2008
33. Maciel, A.; Boulic, R.; Thalmann, D. Efficient Collision Detection within Deforming Spherical Sliding Contact. IEEE Transactions on Visualization and Computer Graphics; IEEE Computer Society; 2007. p. 518-529.
34. Barenblatt, GI. Scaling, Self-similarity, and Intermediate Asymptotics: Dimensional Analysis and Intermediate Asymptotics. Cambridge University Press; 1996.
35. Lim YJ, Deo D, Singh TP, Jones DB, De S. *In situ* measurement and modeling of biomechanical response of human cadaveric soft tissues for use in physics-based laparoscopic surgical simulation. Surgical Endoscopy. (online first).
36. Sankaranarayanan G, Sreekanth A, Lin H, Mulcare M, Zhang L, Derevianko A, Lim R, Fobert D, Cao C, Jones DB, De S. Preliminary face and construct validation study of a virtual basic laparoscopic skill trainer. Journal of Laparoendoscopic & Advanced Surgical Techniques. (submitted).

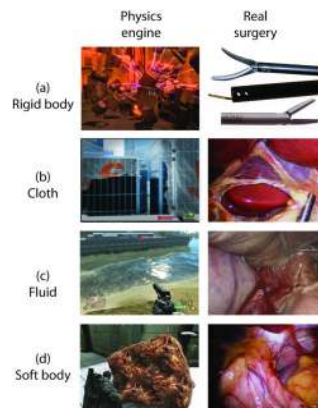


FIGURE 1.

Entities available in PhysX match many surgical entities, such as: (a) rigid bodies may be used for modeling surgical instruments; (b) cloth for membranous tissue; (c) fluids for blood and (d) soft bodies for organs.

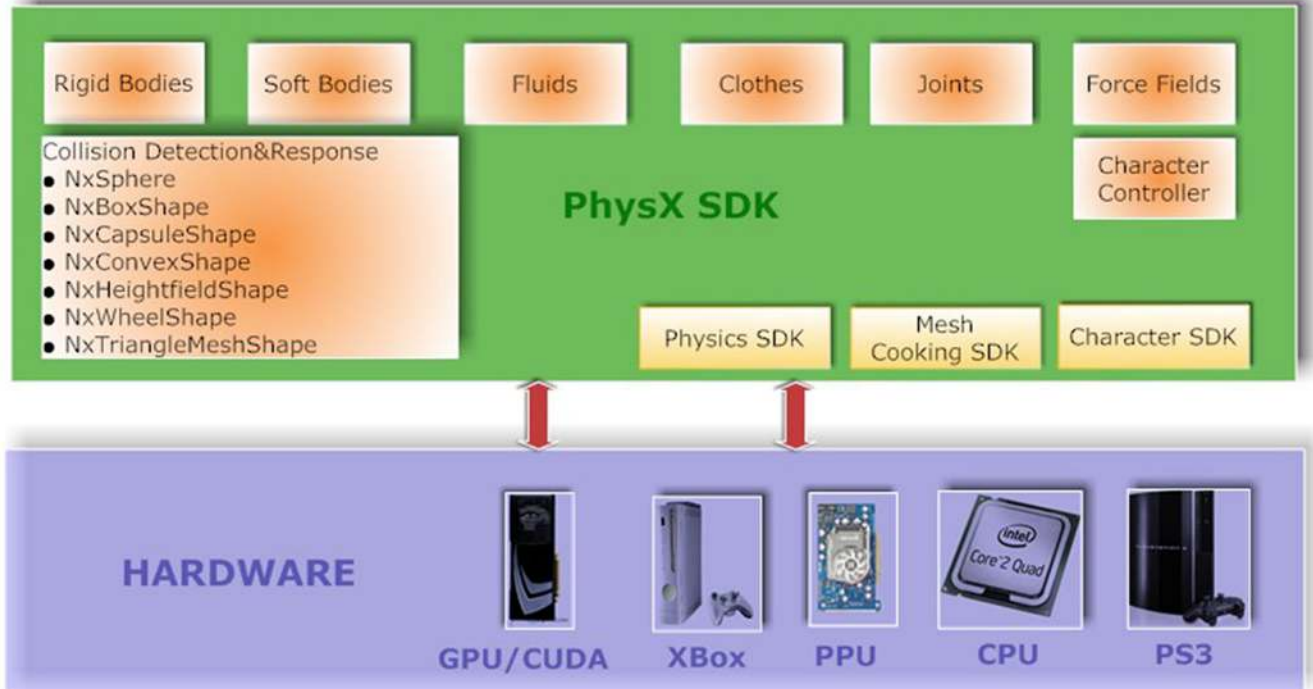


FIGURE 2.
The PhysX architecture.

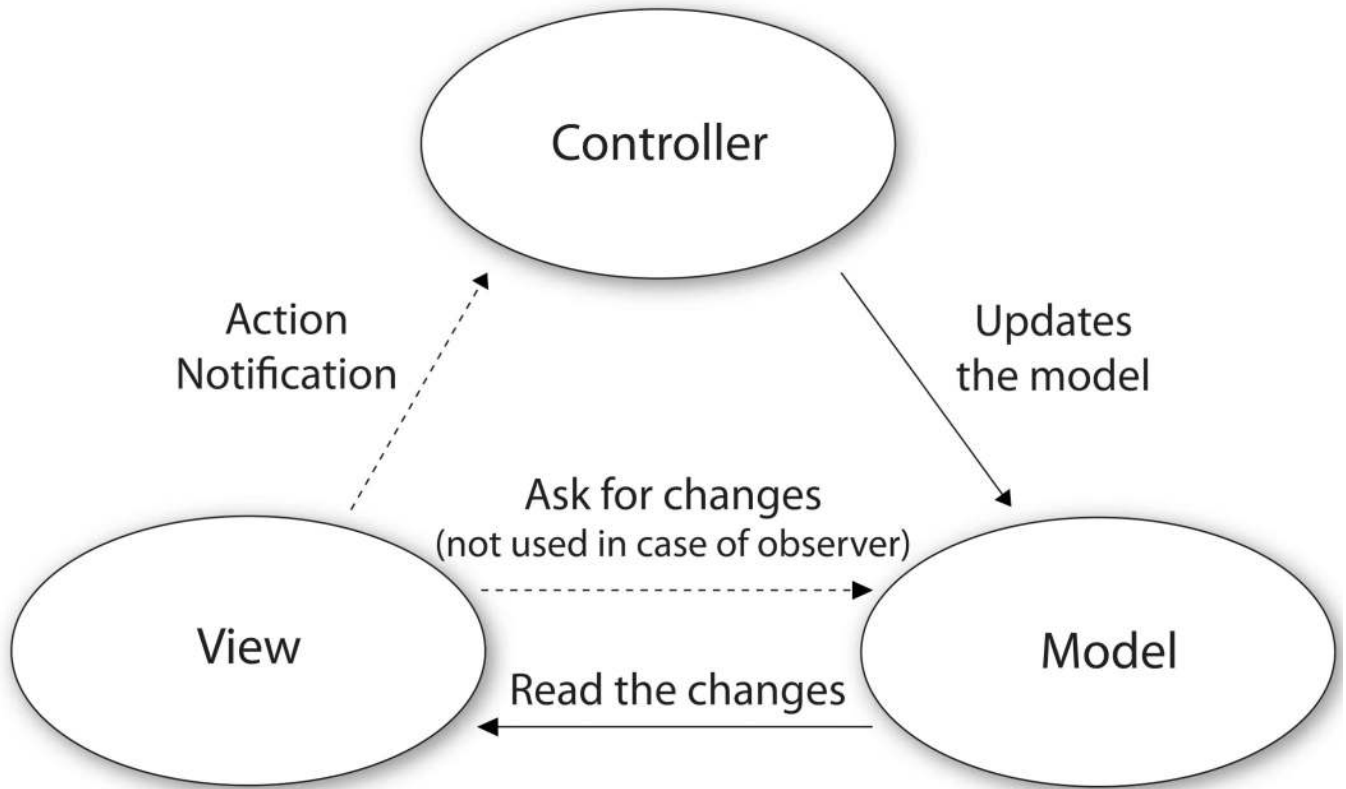


FIGURE 3.
The Model-View-Controller (MVC) pattern.

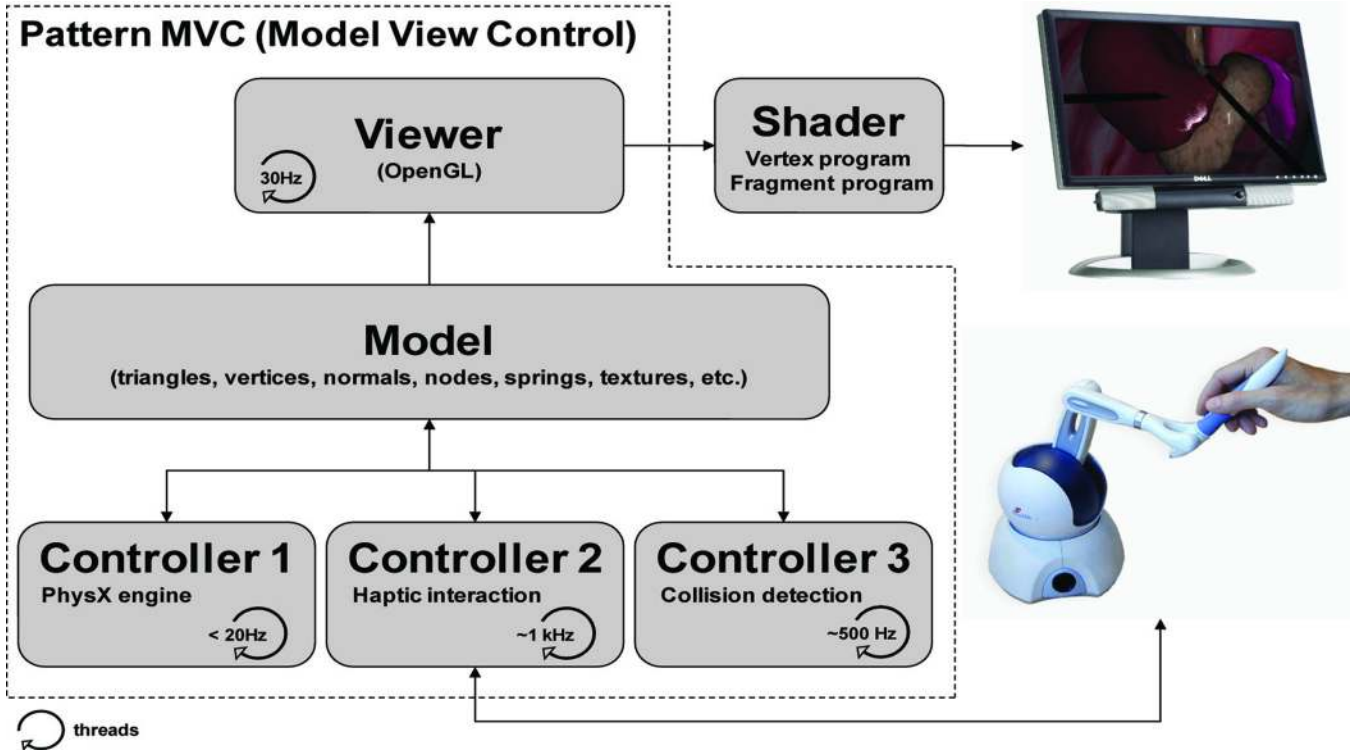


FIGURE 4. The extended MVC framework proposed to deal with different update rates required for collision detection, physics-based interaction, visual and haptic rendering on the same model data structure for surgery simulation.

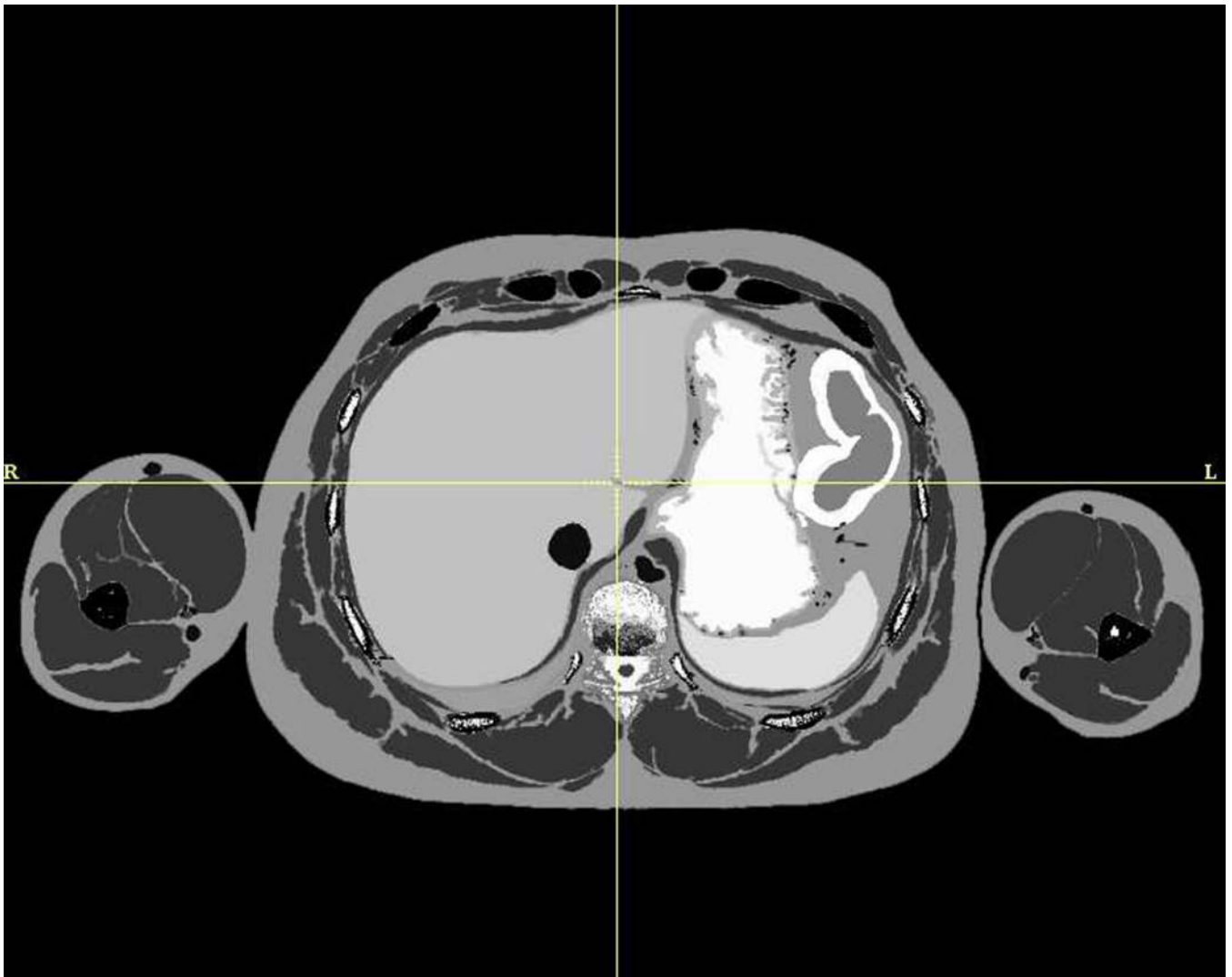


FIGURE 5.
Example slice from the VIP-Man dataset.

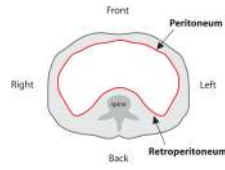


FIGURE 6.
Outline of the peritoneum.

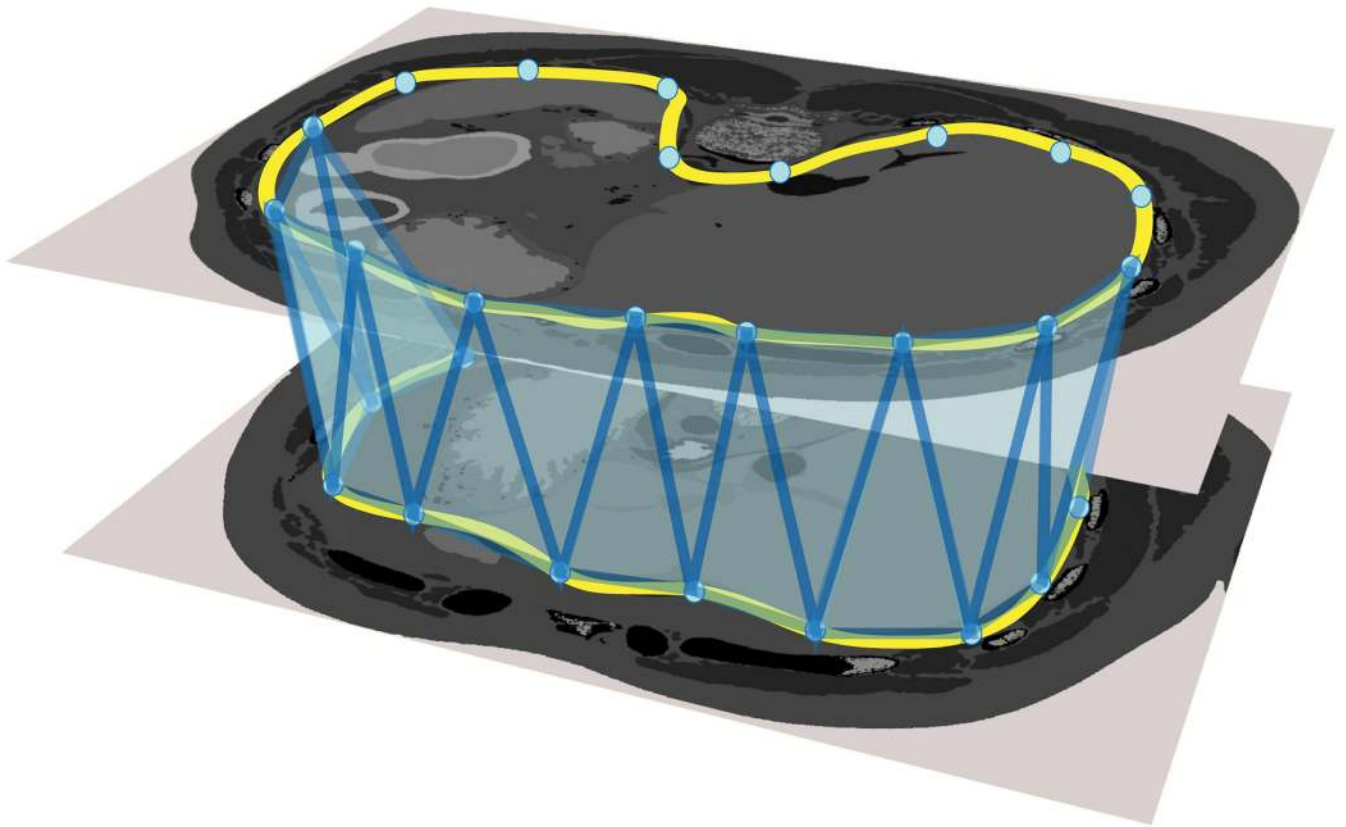


FIGURE 7. Schematic of the peritoneum reconstruction from image slices. Outlines on each pair of consecutive slices have their nodes connected to form triangles.

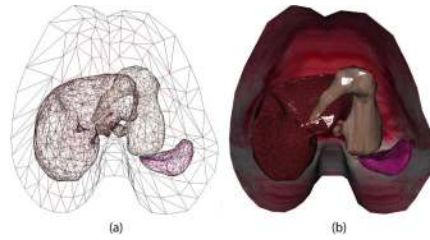


FIGURE 8. Bottom view of the peritoneum model and interior organs in wireframe (a) and textured (b).



FIGURE 9. The Allergan LAP-BAND[®] system (a) and our model (b). The thick part of the band is stiffer than the long part and the two parts are connected by a distance joint.

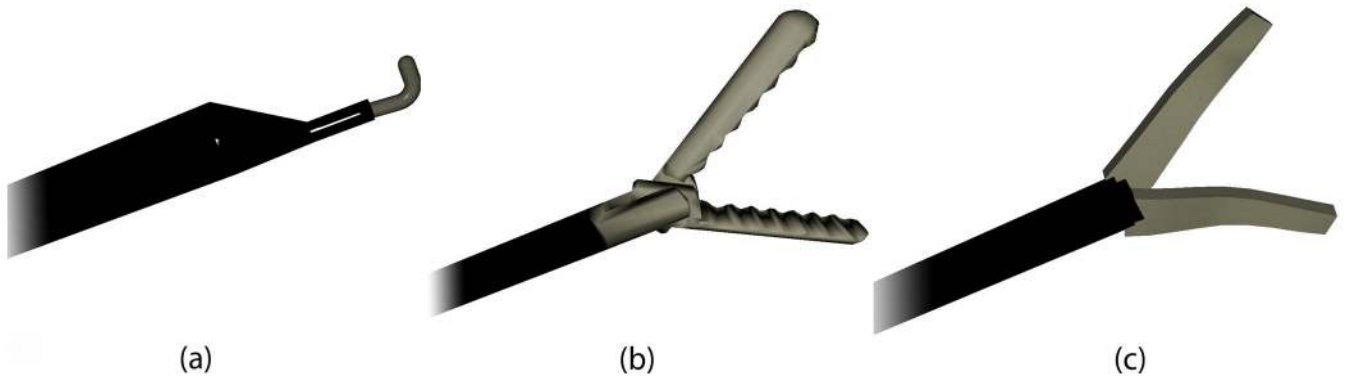
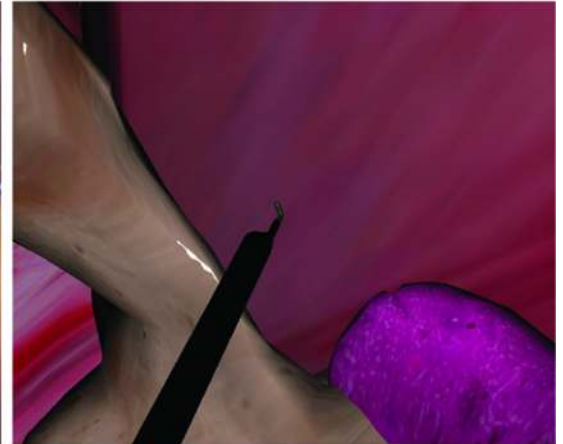
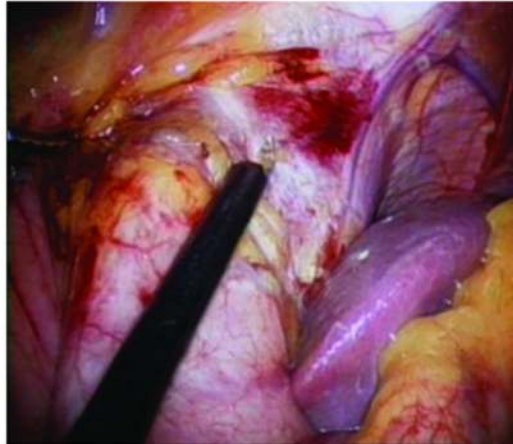


FIGURE 10. Models of the laparoscopic instruments: (a) hook cautery, (b) grasper and (c) scissors.

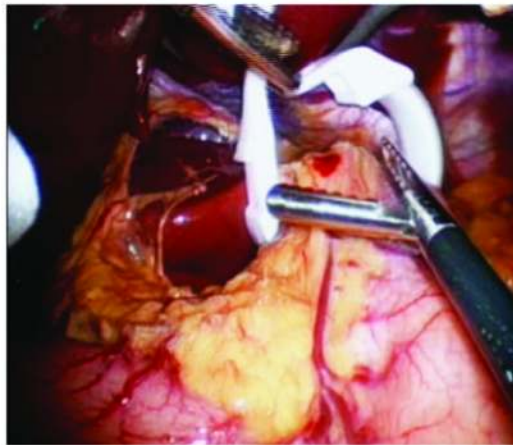
Actual LAGB

Simulated LAGB

(a)
Dissection



(b)
Band
placement

**FIGURE 11.**

Relating actual surgery video frames with screenshots of the simulator. (a) Illustrates the tissue dissection (phase 2) and (b) illustrates the band placement (phase 3). Not all anatomical structures are shown in the current simulated LAGB. Further developments will include fat, ligaments and other important anatomical landmarks.

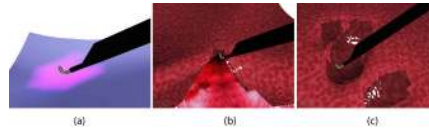


FIGURE 12.

Electrosurgery is used to cut the tissue. In (a) the color mapping shows the temperature distribution, (b) shows the result of cutting a membranous tissue and (c) shows the result of cautery applied to an organ.



FIGURE 13.
Hardware setup of the simulator.

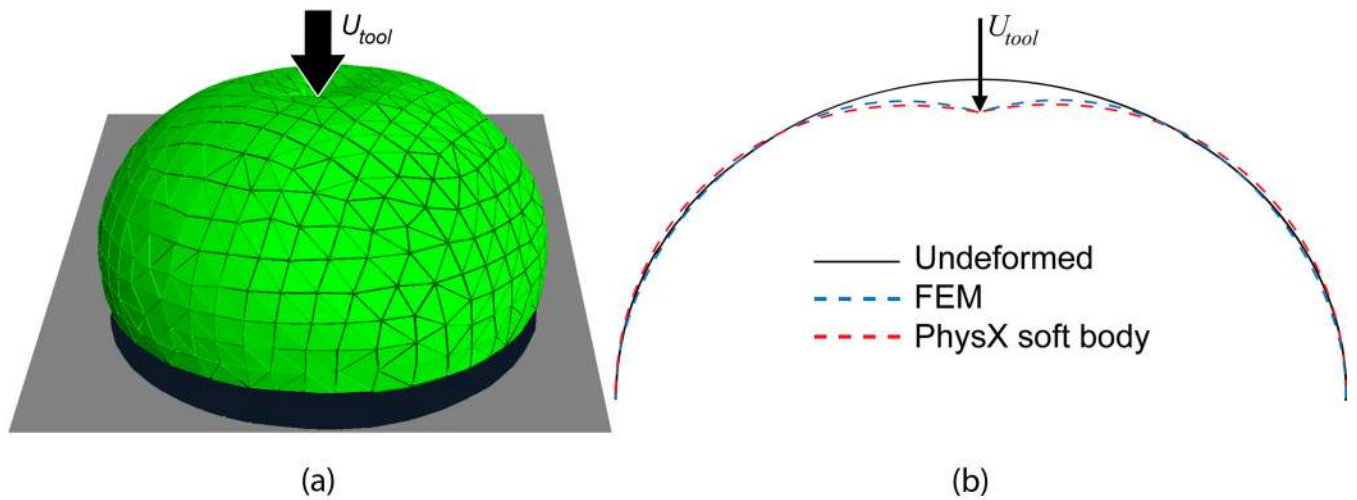


FIGURE 14.

The hemisphere problem. In (a) the hemisphere as modeled in PhysX with 3901 tetrahedra and the applied indentation. In (b) the resulting profiles for PhysX and FEM deformation as well as the undeformed profile are shown.

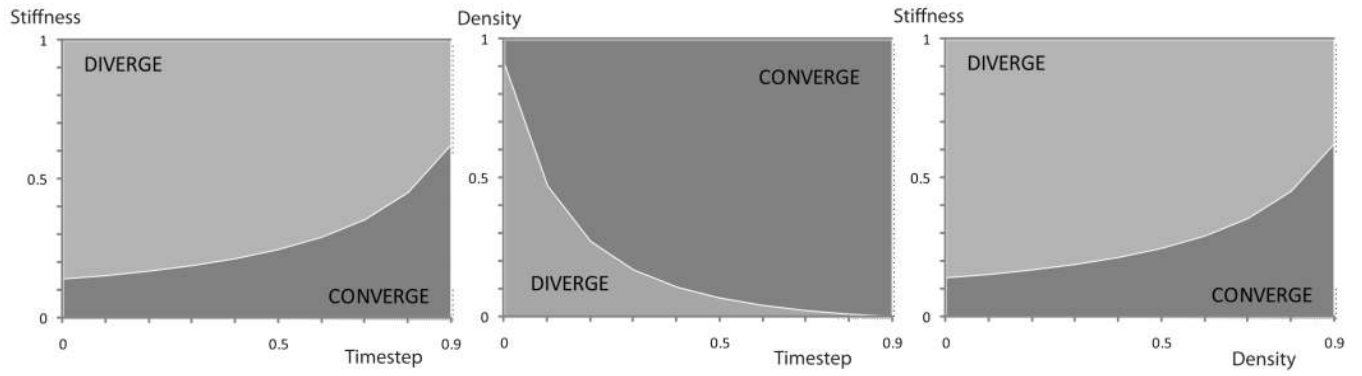


FIGURE 15. Normalized relations between some physical parameters. As a higher stiffness causes the numerical integration to diverge, higher densities make it more stable. However, increasing the number of time steps improves stability but increases computational time.

TABLE 1

Some tissues and their corresponding labels in the VIP-Man image files.

| Organ | Label |
|------------------|--------------|
| Esophagus | 12 |
| Esophagus lumen | 58 |
| Esophagus mucosa | 59 |
| Fat | 29 |
| Liver | 35 |
| Spleen | 40 |
| Stomach wall | 33 |
| Stomach content | 48 |
| Stomach mucosa | 52 |