Published in final edited form as:
Int J Comput Assist Radiol Surg. 2011 July ; 6(4): 457–471. doi:10.1007/s11548-010-0527-3.

Surgical model-view-controller simulation software framework for local and collaborative applications

Anderson Maciel,
Universidade Federal do Rio Grande do Sul, Instituto de Informática, Caixa Postal 15064, 91501-970, Porto Alegre, RS, Brazil

Ganesh Sankaranarayanan,
Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY 12180, USA

Tansel Halic,
Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY 12180, USA

Venkata Sreekanth Arikatla,
Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY 12180, USA

Zhonghua Lu, and
Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY 12180, USA

Suvranu De
Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY 12180, USA

Anderson Maciel: amaciel@inf.ufrgs.br; Ganesh Sankaranarayanan: sankag@rpi.edu; Tansel Halic: halict@rpi.edu; Venkata Sreekanth Arikatla: arikav@rpi.edu; Zhonghua Lu: luz4@rpi.edu; Suvranu De: des@rpi.edu

Abstract

Purpose—Surgical simulations require haptic interactions and collaboration in a shared virtual environment. A software framework for decoupled surgical simulation based on a multi-controller and multi-viewer model-view-controller (MVC) pattern was developed and tested.

Methods—A software framework for multimodal virtual environments was designed, supporting both visual interactions and haptic feedback while providing developers with an integration tool for heterogeneous architectures maintaining high performance, simplicity of implementation, and straightforward extension. The framework uses decoupled simulation with updates of over 1,000 Hz for haptics and accommodates networked simulation with delays of over 1,000 ms without performance penalty.

Results—The simulation software framework was implemented and was used to support the design of virtual reality-based surgery simulation systems. The framework supports the high level of complexity of such applications and the fast response required for interaction with haptics. The efficacy of the framework was tested by implementation of a minimally invasive surgery simulator.

Conclusion—A decoupled simulation approach can be implemented as a framework to handle simultaneous processes of the system at the various frame rates each process requires. The framework was successfully used to develop collaborative virtual environments (VEs) involving geographically distributed users connected through a network, with the results comparable to VEs for local users.

Correspondence to: Anderson Maciel, amaciel@inf.ufrgs.brandi.maciel@gmail.com.

Electronic supplementary material The online version of this article (doi:10.1007/s11548-010-0527-3) contains supplementary material, which is available to authorized users.

Keywords

Surgical simulation; Virtual environments; Physics-based simulation; User interfaces-Haptic I/O; Software framework

Introduction

Over the years, virtual environments (VE) have become increasingly demanding in terms of realism, complexity and interactivity. For example, a typical VE for surgery 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 [16]. To achieve a certain degree of realism in such complex surgical interactions, the simulated virtual environments need to support heterogeneous scenes composed of different states of matter (solids, liquids and gases); complex geometry and material properties of objects within the scene; dynamic and real-time **interaction** between virtual objects and tools physically manipulated by the user; and multimodal (visual and haptic) rendering of the results to the user. It is clear that it is not possible to satisfy all the requirements using a single modeling technique. It is therefore necessary to use heterogeneous techniques, each optimized for one or more of the tasks.

Shared virtual environments [6], which allow remotely located users to interact within the VE, bring an additional layer of complexity. For example, a *Networked Haptic Surgical Environment* (NHSE) allows multiple users to individually or simultaneously train on various surgical procedures [12,32]. See an example in (Fig. 15). Such an NHSE consists of one or more massively parallel high-performance computing platforms simulating highly accurate anatomical models in scenarios involving complex tool–tissue interactions and deformations. The clients are connected to the server via Internet link using a high-end PC or workstation. This allows clients located at disparate geographical locations to run models of high computational complexity on commodity hardware. In addition, oftentimes, qualified experts may not be available at the same geographical locations as the trainee, and the NHSE provides the perfect platform for collaborative training/telementoring. Moreover, NHSE could be used as a platform for training surgeons for tele-surgery applications [9,33] and [34].

Hence, the development of a systematic framework that allows the separation of simulation, visualization and interaction processes is absolutely essential. However, first it is important to differentiate between software frameworks and software libraries. Within a framework—unlike in a library—the overall flow of control of the program is dictated by the framework and not by the caller [25]. Hence, software frameworks contain a considerable amount of management and utility code in order to support the development of user applications. According to Pree [24], software frameworks are a combination of *frozen spots* and *hot spots*. *Frozen spots* define the overall architecture of a software system, i.e., its basic components and their relationships. These remain unchanged (frozen) at any instance of the application framework. *Hot spots*, in turn, represent the parts where the programmers who use the framework add their own code to include their own project-specific functionality. Generally, frameworks focus on specific problem domains including artistic drawing [31], visualization [28], music composition, computer-aided design, compilers for different programming languages and target machines [1], financial modeling applications [4], decision support systems [9], media playback and authoring, web applications, middleware, etc.

While a variety of software frameworks are currently available for multimodal VEs, most fail in providing real-time physics-based interactions with scenarios composed of complex realistic models while, at the same time, allowing haptic interactions. Most frameworks focus on separating the graphical user interface from the underlying software layers. XVR [26] or VHD++ [23], for example, support haptics in VR environments where the user can interact with rigid objects using constraint effects. However, when haptic interaction is desirable in order to manipulate complex soft deformable bodies, as in simulated surgery, there is a bottleneck since the computational power is being consumed by costly numerical processing related to physics-based models and interactions.

Classical software design patterns and models are implemented in state-of-the-art frameworks. Design patterns are meta-structures for software development which define templates to build solutions for sets of problems with common characteristics. They are called “classical” because they are stable standards published in software engineering books and applied by most programmers. Frameworks combining any number of design patterns are now available as application templates. Although such frameworks do achieve the goals of providing higher productivity and shorter time-to-market for the development of the intended applications, they are hardly perfect and there may be significant problems in designing, learning and using these frameworks. The problems in design are related to fitting. As the frameworks are as general as possible to fit the greatest number of possible applications, they end up not fitting tightly to any application. In practice, this means that the designer has to end up by modifying their ideas to fit the framework which may lead to an inadequate solution. On the other hand, as a framework is as specialized as possible to fit a specific application, it runs the risk of being the application itself, leaving not much of a room for the programmer to change. Problems in learning a framework arise when the framework changes the development paradigm substantially. Of course, anything new will take some time to be apprehended, but the learning curve could be much shorter if a number of known landmarks and *a priori* were present. In practice, if developers spend more time learning the framework than the time they save by using the framework, the framework will probably not be used.

In addition, in the constantly changing scenario where different interaction techniques devices and applications become obsolete in quick succession, such classical solutions are not sufficient. Even in the recent past, this problem was barely noticeable due to strong hardware limitations. However, the appearance of multiprocessor GPUs and multiple core CPUs have ushered in an era of unprecedented computational power for commodity desktop and laptops and these limitations can no longer be neglected.

An example of a currently developed open source framework is SOFA, which stands for ‘Simulation Open Framework Architecture’. This has an emphasis on medical simulation [2]. It currently integrates, in the same environment, a variety of different algorithms: springs, co-rotational FEM, free form deformation grids, implicit and explicit solvers, and several collision detection methods. SOFA has shown plausible results with non-realistic deformable models [27] and has a recent implementation to specifically exploit parallelism in multicore hardware architectures [13]. Limitations of SOFA include the lack of support for haptic feedback. Another limitation is the difficulty to allow topological changes in the model. This is a consequence of the multi-model representation used in SOFA. Although it is presented as an advantage, it makes the task of maintaining coherence among the different models complex and time consuming. Other frameworks for medical simulation usually propose severe modeling simplifications which go as far as limiting the interactions to point-based simulation [35].

Real-time physics engines, originally developed for game development, offer another alternative. Some of these engines are available commercially, others have closed source or are under limited license such as Havok, PhysX 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. For more information about their simulation capabilities, we encourage the reader to look into the work of [29] and [5]. 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 complex VEs such as surgical scenarios. This can be due to the lack of a framework featuring a well-balanced mechanism for exploration of modern hardware resources. Marks, Windsor, and Wünsche [22] investigated some 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, Unreal Engine, and Source Engine were chosen for further evaluation. These engines were evaluated in terms of editing, content and gameplay. More specifically for the surgical case, their assessment focused on the maintainability of the surgical task. They analyzed 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, the quality of physics-based interaction. The conclusion is that game engines actually provide a great set of features that help the development of simulated training applications. Such features include graphics capabilities, audio and networking, and also a good use of the underlying parallel hardware. The major drawback when using current game engines for surgical simulation is that modeling highly mathematical physics models is not possible yet. Simplified physics-based models can be adapted, but they do not provide the rich details necessary for realistic and effective surgery simulation [3]. Any more complex physics models in game engines will have to compete for CPU and GPU time with other components of the engine such as graphics and management. Hence, current game engines do not provide the reliability required for real-time medical simulation.

In summary, while several frameworks exist, none is capable of supporting the level of complexity that is necessary to develop complex, high fidelity applications such as surgical simulations that allow haptic interactions as well as collaborative features necessary for shared virtual environments. In this paper, we discuss the development of a powerful, unified framework based on the well-known model-view-controller (MVC) pattern [10] for multimodal VEs that supports physics-based interactions including force feedback and allows networking. The unique advantage of this framework over existing ones is a schema which combines the well-known MVC pattern with support for parallel hardware and multimodal interaction with force feedback. The development of this framework will be presented in the context of a case study—physics-based surgery simulation with haptics. While surgery simulation is, of course, not the only application, it provides some of the toughest challenges as its purpose is to train surgeons in the skills necessary in their profession and improve operating room outcomes [4].

In the section “MVC pattern for VE”, we discuss the MVC pattern and how this might be used to develop an efficient framework for multimodal VEs with haptic feedback. In section “Some applications”, we present case studies applying this framework to both local and remote surgery simulation. In “Concluding remarks”, we discuss some capabilities of the present framework and our plans for the future.

MVC pattern for VE

The model-view-controller (MVC) pattern

Our framework is based on the model-view-controller (MVC) pattern [10]. 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. See Fig. 1 for a schematic of the 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. 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, the Model holds geometric and material information in the form of a scene graph, 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.

In this paradigm, Viewers and Controllers have a stable application program interface with the Model. This allows a developer to replace a Controller or a Viewer by another as long as they respect the interface. Essentially, as a Controller reads information from the model and writes modified information to it, any custom or third-party software can be integrated in the framework as a Controller. This includes open source or commercial libraries, engines, plug-ins and toolkits, as well as in-house procedures. If new interaction devices are required for new applications, they can be also integrated into the framework as a new Controller. Analogously, a viewer just reads data from the Model and displays it. If new rendering methods or new display devices are required, they can be implemented as a new viewer.

Design and implementation

Decoupled simulations have been used to manage 20-Hz display frequency with modification of the viewpoint while the simulation itself was performed at a lower rate [30]. Our framework is designed in such a way that multiple execution threads run in parallel and concurrently at different rates and in separate cores on the latest CPUs allowing two-ways interaction, not only visualization. Our decoupled strategy permits, for example, transparent haptic computation at over 1000 Hz in a Controller while a custom physics-based simulation engine runs at lower frequencies, e.g., below 400 Hz, in another. At the same time, graphics rendering is achieved at 30 Hz. Figure 1 presents a schematic of our framework. In the following sections, we discuss the various modules.

The model module—The Model module, as in the MVC pattern, is a data structure which comprises a set of classes necessary to describe the VE geometry, status, and physical properties (Fig. 2). As stated in the previous section, the Model does not define behavior directly.

Model in our framework defines the primitives required by OpenGL for rendering and the physics attributes of materials necessary for simulation. It also includes visual attributes such as textures, and many intermediary structures that can be used for a number of Controllers such as boundary volume trees and topological structures. In our case study, a scene is composed of organs and surgical instruments which are defined by triangle meshes. Mechanical properties defining the stiffness of a deformable mesh and the location of a rigid

mesh are both part of the Model, but the ability to deform or displace these elements is defined in a Controller called the Simulator (see section “The simulator controller” below).

To extend the Model, new classes can be created specializing existing classes. One example is to extend the functionality of the class Mesh with MeshFEM (for a finite elements method implementation) or MeshMSD (for a mass-spring damper implementation).

The view module—Our View module uses OpenGL and GLSL to render a frame buffer with full multi-textured color operations per vertex and per pixel (Fig. 3). It can combine information from up to four textures loaded in the Model and light information to provide final color on the screen. The textures are currently used for color maps, normal maps and edit maps. For instance, in surgical simulation environments, color maps may be generated from photographs of real organs, while normal maps are used to provide bump and relief maps, and edit maps to represent surgical alterations on the tissues such as burning. The textures are combined, blended, and rasterized in the fragment shader, in such a way that the realism algorithms are executed in the GPU. This produces outstanding visual realism without undue demand for CPU resources.

Our View module also allows us to render the scene from two or more simultaneous points of view, enabling stereo visualization. As the modern GPUs allow for a large frame buffer, we render all points of view on the same frame buffer and allow the operating system or the graphics card driver to control which part of the viewport is directed to which display hardware. To demonstrate this capability, we have used a Planar Stereo display with two polarized LCD monitors and polarized glasses (see Fig. 9) for stereo visualization.

For non-conventional displays, like some types of head mounted displays or tiled displays, a similar strategy may be used. Of course, as newer display systems become available, additional viewer modules would be necessary.

The haptics controller—The Haptics Controller enables the communication of the Model with the force-feedback device (Fig. 4). This Controller is not responsible for calculating the forces itself. The interaction forces are calculated by the Simulator and Collision Detection Controllers and stored in the Model. The haptics Controller fetches the current state of the haptic device which includes positions and orientations of the end effecter and device buttons to the Model. It also sends the computed force from the Model to the device. The haptics Controller is a generic device controller but customizable to commercially available haptic devices. Device-specific implementation is achieved by integrating corresponding device API functions to the haptics Controller functions. Hardware level communication to the devices is enabled by the device specific drivers supplied by the manufacturer. Currently, the haptics Controller supports Sensable PHANToM devices through OpenHapitcs API, and NOVINT Falcon through Novint API. The haptics Controller API has higher level functions that can transform the various device positions and orientations from the device coordinates to the simulation world coordinates, before updating the Model with the new information. Such conversions are often necessary to register 3D spaces and to apply scaling between worlds (virtual and real worlds, for example).

The asynchronous design of the haptics Controller makes it possible to run this Controller at a frequency of more than 10 kHz, which is much higher than the necessary to maintain smooth haptic interactions.

The collision detection controller—Collision detection is often the bottleneck in an interactive VE simulation. Hence, the collision detection code must run with high priority

and maximum resources. The collision detection code must also be highly optimized. Various collision detection algorithms are possible, each with its own set of merits and limitation [17]. As our framework is intended to be used in interactive multimodal VE's, providing access to different collision detection methods will greatly increase its flexibility. We have included in our framework a number of classical collision detection algorithms such as OBB [11], z-buffer, dynamic point [19], PQP [15], and the collision module available as part of the PhysX library.

It is expected that the virtual environment will be composed of heterogeneous objects. Hence, it might be necessary to apply multiple collision detection algorithms in the same scenario. For example, in surgical simulation, a line to mesh collision may be necessary for tool to organ collision and OBB-based collision for detecting self collision of the deforming organs. The MVC architecture allows such multiple collision detection algorithms to run simultaneously as each collision method is a separate Controller which is applicable to two or more objects in the scene, hence more than one collision detection algorithm may be implemented in the same Controller.

Each Controller (Fig. 5) is built as a separate thread object that can communicate directly with the Model. It also contains method-specific data structures to handle collisions. The Controller reads the updated information from the Model, checks for collision, and updates the data structure with collision information. In this data structure (Model), individual collision objects are instantiated. Collision objects are stored in a list of collisions which starts as 'void' and is filled with collision elements as collisions are detected. Each collision object in the list, in turn, points to the primitives that are colliding (triangles, tetrahedrons, lines, boxes, spheres and so on, depending on the collision detection method) and other necessary information for collision response, such as distance of penetration, direction of contact, penalty forces, and velocity constraints.

The Viewer and other Controllers can access this collision list directly from the Model to perform necessary actions. For example, a physics Controller can apply deformation to soft bodies that have just collided or, in the case of the Viewer, the colliding triangles may be shaded in red for display.

The simulator controller—This Controller applies physical rules to elements in the Model to modify their position, velocity, acceleration, and other physical attributes (Fig. 6). A multimodal virtual environment must also support multiple physical interactions, e.g., it should be possible to simulate rigid bodies, deformable soft objects, fluid flows, smoke generation, electrothermal and thermo-mechanical interactions. Each of these physical processes may be simulated in multiple ways. For instance, soft bodies may be simulated using mass-springs, finite elements, point associated finite fields [7] or boundary elements, whereas particle systems or smoothed particle hydrodynamics methods may be used for the simulation of fluid flows.

In our framework, the Simulator implements various physical interactions using some of these techniques. The Simulator communicates with the Model through which inputs—such as geometry, positions, velocities, accelerations and torques, collision constraints and material properties—are passed to the Controller. These attributes are then updated by the Simulator using implicit or explicit integration according to each case. It is known that some modeling strategies provide faster convergence with stiffer models (e.g., FEM) while others converge faster with softer models (e.g., mass-spring). For this reason, we used a wide range of empiric values for the stiffness of both the models and the force feedback. Although no formal evaluation has been performed to date to quantitatively assess user perception of the model behavior upon interaction, we worked this aspect qualitatively. Our prototyping

methodology included frequent interviews with experienced surgeons who tested the system and provided feedback which allowed us to tune model and response stiffness.

In the scenario of virtual surgery, some objects in the simulation scene are kinematically controlled by the interaction device (surgical instruments), some are defined by rigid-bodies dynamics (needles) and yet others are controlled by soft-bodies dynamics (organs). In simulating organs, some can be implemented as mass-spring systems while others as finite elements or other equivalent models. While each physics-based method in the Simulator Controller is responsible only for its own class of objects, the Collision Detection Controller provides transparent interaction between them. In the framework, several types of simulation methods have been implemented including mass-spring and FEM. Moreover, a physics engine (PhysX) has also been included in the simulator as a separate Controller [19].

The input controller—Various input devices enable interaction between the user/s and the VE. These devices can be as simple as keyboards and mice to more complex devices such as joysticks, foot pedals, and head tracking devices. The Input Controller (Fig. 7) handles all specialized inputs and updates the Model with the information. The haptic devices, however, are specialized input/output devices due to their high update requirements (1000 Hz) and are implemented as a separate Controller as described in section “The haptics controller”.

In our framework, the Input Controller has two different types of input processing. The Qt peripheral management routines which run as Qt event threads are used to process keyboard and mouse inputs. The same routine can then be used to read the status of a foot pedal as it emulates the mouse buttons. The other type of input processing is for information from sensors connected to the VE interface. For example, these can be the information from a motion tracking device or in the case of surgical simulation, output from the encoders detecting tool movements. Currently, this is supported by the ADU 100 USB Interface from Ontrak Control Systems Inc. The support for other input devices can be implemented using a straightforward extension of the existing module.

The network controller—Our framework has also built-in network capabilities for collaborative surgical and telementoring simulations. It could also be used for simulating tele-surgery. It supports peer-to-peer, client–server, and hybrid network architectures. The Network Controller (Fig. 8) in the framework handles all UDP and TCP/IP packet communications. It is fully capable of handling lost and out-of-sequence packets.

The Network Controller enables remote users to collaborate and share the resources of the framework. The Controller runs asynchronously in a separate thread. It consists of a socket layer with an open UDP and TCP/IP port for communication. The UDP socket is used for haptic data communication since it requires a very high update (usually 1000 Hz for haptic applications). The TCP/IP socket is used for client–server communications.

Whenever there is a packet ready to read in the socket, the appropriate TCP or UDP packet handler is called by the packet callback function. Each UDP packet carries a unique sequence number and a checksum so that the out-of-sequence and corrupt packets are detected and dropped. The packet handler then updates the packet data structure with the new network data obtained from other users. This data are then queried by other Controllers in the framework for interaction. When there is no new packet information, the previous packet data are stored until this information can be updated; essentially this represents a zeroth order hold. In the context of the simulated surgery, typical information that is passed between the networked users include the quaternion for the position and orientation of the

surgical instruments, grabbed or contacted node on the model, time stamp, packet sequence number, checksum, and other relevant information for particular scenarios.

Some applications

In this section, we demonstrate the capabilities of framework through some application scenarios. The first two applications are surgery simulation scenarios. The Virtual Basic Laparoscopy Skill Trainer (VBLaST™) is a virtual box for trainee surgeons to practice and evaluate their skills [20]. The Laparoscopic Adjustable Gastric Banding (LAGB) procedure is a minimally invasive surgery where a ring is placed around the stomach to treat cases of severe obesity. The third application is a Networked Haptic Surgical Environment, where the effect of network latency on deformable object has been investigated. We present these applications to show that many problems can be solved in the middleware layer of the framework. Particularly, we demonstrate the capability of the framework to simulate different types of dynamic objects such as rigid bodies, cloth, soft bodies, and various types of collision interactions.

A virtual basic laparoscopic skill trainer (VBLaST™)

Surgical skill training is a long and tedious process of acquiring fine motor skills. The Fundamentals of Laparoscopic Surgery (FLS) training tool box [14] consists of a box covered by an opaque membrane through which trocars are placed on either side of a laparoscope connected to a video monitor. Inside the box, five pre-manufactured tasks including peg transfer, pattern cutting, ligating a loop, and suturing can be performed.

To overcome potential drawbacks of such mechanical toolbox systems, we have developed a Virtual Basic Laparoscopic Skill Trainer (VBLaST™) whereby tasks, such as the ones available in the FLS system, may be performed on the computer.

A stereoscopic visiohaptic workstation has been developed including two Phantom Omni™ force-feedback devices and a 3D display interface from Planar Systems, Inc (see Fig. 9). During the simulation, the participants use the Phantom Omnis to control the virtual surgical instruments in order to interact with the virtual training materials. On the virtual side, the instruments represented in the system are tissue graspers, suture graspers, scissors, and the ligating loop.

Consistent with the FLS system, we have developed the following tasks within VBLaST™ (see Fig. 10):

Peg transfer—Each of the six virtual rings may be lifted from a virtual pegboard with the left hand, transferred to the right hand, and placed on another pegboard.

Pattern cutting—A 4-cm-diameter pre-marked circular pattern may be cut out of a 10 cm × 10 cm piece of virtual gauze suspended between alligator clips.

Ligating loop—A 3-dimensional tubular structure is presented in space. Using bimanual manipulation, a virtual loop is securely fashioned about a pre-drawn line on the tubular structure.

Suturing—A virtual suture is tied using either an intracorporeal or extra-corporeal knot, using 3-dimensional bimanual manipulation with a curved needle.

Collision detection and dynamic response computation in a virtual environment are necessary so that one can pick, move, or even cut objects at the same time that effort

responses are generated. An efficient dynamic point algorithm is used for line-based collision detection and response. Readers can refer to [19] for more details.

Various physics-based algorithms are used for interaction with the various virtual objects. For peg transfer, a rigid body simulation was used as the Controller. However, for both pattern cutting and ligation loop, a mass-spring simulator was used. For suturing, a FEM simulator was used for deformation calculation while the suture thread and the needle were simulated using 1D mass-spring and rigid body dynamics, respectively.

The peg-transfer task relies on implementing correct rigid body dynamics. The rigid body Controller in the simulator uses an Euler explicit integrator to compute linear and angular positions from linear momentums and angular momentums acting on the rings during the interaction with the tools. As only rigid dynamic and static bodies interact in this simulation, for collision detection, the Proximity Query Package or PQP [15] has been used. An impulse-based collision response is applied to the rings to prevent interpenetration and ensure physically correct interactions with the environment. In addition, bimanual manipulation of the rings has been implemented in order to give correct force feedback to both the hands as the rings are transferred from one hand to the other.

For pattern cutting, ligation loop and suture thread a mass-spring simulator has been used to compute and display deformations. The mass-spring simulator can be a 3D surface (tube for the ligation loop), 2D surface (cloth for pattern cutting), or a 1D model (thread for suture). It also has damping elements for added stability. An Euler integrator has been used to compute new positions of the mass nodes. Penalty forces calculated based on collisions have been used to handle contact with deformable objects.

In the mass-spring simulator, picking of the object has been enabled by checking the condition when the two jaws of the tool were closing and finding the surface triangles that are in contact with both the upper and lower jaws of the tool. The contact triangle positions are then explicitly set to move to follow the tool. The rest of the nodes react and move accordingly due to the forces exerted on them from the springs connecting each of the mass nodes of the triangles. In pattern cutting, this condition was used to remove the springs around the triangle to simulate opening of the gauze while cutting.

For the suture thread simulation using 1D mass-springs, an impulse-based collision response is computed for both tool-to-thread and thread-to-thread self collisions. This enables the thread to be tied in a knot using intracorporeal suturing techniques.

For the suturing task, a 3D linear quasi-static FEM simulator has been used for the deformation of the suturing base object. During simulation, the user is allowed to interact with any of the nodes belonging to the surface of the model. The changing position of the interaction tip is read from the device to estimate the displacement boundary condition of the interacted node. In quasi-static analysis, the displacement boundary conditions are applied continuously at each cycle to mimic continuous interaction.

Tetrahedral elements are used to discretize the deforming object. A straightforward implementation of the finite element procedure would not result in real-time performance. Hence, a pre-computation-based algorithm has been implemented. In this technique, the inverse of the stiffness matrix of the deforming model is pre-computed after application of the fixed boundary conditions. At run-time, the pre-computed inverse is used to compute the deformation field when the user interacts with the model.

To simulate the action of suturing of this finite element model, a constraint is applied at the interaction point of the needle and the model and the corresponding displacement field is

computed using the FEM simulator. A similar procedure has been applied to model for the suture thread interaction as it is constrained to follow the needle through the insertion and exit points in the body.

A Laparoscopic Adjustable Gastric Banding (LAGB) simulator

The Laparoscopic Adjustable Gastric Banding 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 sensation of satiety. The motivation for this specific surgical procedure is the high cost in training surgeons and the increasing demand for it.

We simulate the pars flaccida technique for the lap-band placement, which involves basically three phases: port placement, dissection, and band placement. As the port placement phase is extracorporeal, we focus our modeling on the remaining two phases.

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, and the gastrohepatic ligament (pars flaccida) is divided and then the gastric fundus and omental fat are retracted. 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.

We modeled a hook cautery (Fig. 11a) as a rigid body and a blunt dissector or grasper (Fig. 11b) as 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 by modeling the temperature increase caused by applying the cautery to the tissue, removing triangles when vaporization temperature is reached. Figure 12 shows the result of cutting different types of tissues using the cautery tool. 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 extensively combined custom methods and PhysX capabilities in our framework. The organ deformation is calculated using PhysX, while the graphics rendering is accomplished using our custom shaders. The PhysX Controller uses the *softbody* simulation object for calculating deformations. Internally, PhysX uses volumetric tetrahedral elements and position-based dynamics to update the soft bodies. As soft body-to-soft body collision detection is not efficient in PhysX, the method described by [21] 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.

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. The major steps in this modeling are explained below, with Fig. 13 illustrating these steps.

The instruments used in the previous phase are also used in this phase. Two graspers are used to manipulate the band. 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 [21] but collisions with the instruments are resolved using PhysX. As for picking, when the grasper closes, all colliding triangles are flagged as picked by that instrument in the Model. Then, as the closed grasper moves around, all picked triangles are kinematically manipulated by the PhysX Controller, 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 the physics Controller to smoothly follow the displacements. Flags are reset when the grasper opens to release the picked triangles.

For a thorough view of the LAGB application within the framework and of how it benefits from the framework topology, please refer to Fig. 14.

A networked haptic surgical environment

Collaborative training requires interaction over a network. Due to network latency, performing even a simple surgical task gets complicated. In order to gain vital expertise in performing the surgical tasks over the internet under various communication latencies, a networked simulation environment was built using our framework. The Network Controller of the framework was used to enable network connections that communicated to and from the virtual salve.

The tele-surgery simulation setup shown in Fig. 16 consists of two workstations located inside our laboratory in Troy, NY. WS1 was running the framework which consists of a 6 DOF virtual slave that can interact with the virtual deformable models. A PHANToM Omni haptic device connected to WS1 acted as a master device that controlled the virtual slave manipulator. WS2 was running a packet reflector program that reflected to WS1 the packets sent to it. The positions and orientations from the master device were routed through the Internet using a packet reflector program hosted at our collaborator's site in Seattle, WA. UDP was used for communication between the master and virtual slave device. When there is no delay the UDP data packets were sent to WS2 which is then reflected back to WS1. For time varying delay, the packets were sent to WS2 through reflector in Seattle and reflected back to WS1 through the same packet reflector in Seattle. Note that using a packet reflector, twice the amount of normal latency was simulated while operating in network with latencies. The packets were routed through the Internet 2 gigabit network.

The snapshot from the simulation is shown in Fig. 17. The slave environment consists of a deformable liver and a stomach model placed inside the peritoneum. The virtual slave can be used to interact with the deformable models for grasping, cutting, or electrocautery. Based on the selected packet reflector location, different communication latencies may be simulated for training.

Figure 18 shows the X-axis position tracking of both the master and the virtual slave during a trial with data packets reflected through the server in Seattle. Figure 19 shows the round trip time varying delay between WS1 and WS2 for the packet reflector location in Seattle. The average delay was 150.57 ms. From Fig. 18, one can see that the virtual slave has perfectly tracked the master device while interacting with the deformable models under time-varying communication delay.

Concluding remarks

We have designed and implemented a multimodal simulation framework and tested it with various simulation scenarios involving both local and collaborative interactions. The framework relies on a combination of available libraries, toolkits, physics engines, 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 and physics engines, dedicated libraries and other existing standard or custom software have the potential to improve efficiency in development and quality of the final results in VE applications. The greatest challenge is 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.

We have developed example applications that involve complex realistic scenarios to evaluate the performance and extensibility provided by the framework. For all our applications, the system displays graphics at 60 Hz, corresponding to 30 Hz for each monitor/eye for stereo visualization. Such frequencies provide smooth graphical displays with no flickering. The haptic response is provided at frequencies over 1,000 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 100–4,000 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 time integration methods are simple but require more steps (usually 300–600 steps per second for real time) for convergence, engines using implicit methods are more time consuming per frame but are unconditionally stable. Typically, our simulation processes physics at 10–20 Hz when using implicit integration, without interference in the performance of other modules.

Our team is composed of researchers some of whom are mechanical engineers, some are computer graphics people, and others are medical personnel. Once the core of the framework was developed, most of the team members were able to implement and use their own Controllers and create their own specialization of classes for their intended applications. This underscores the utility of having such a framework in a multidisciplinary research environment.

As a part of future work, we plan to support interaction between objects simulated with various possible simulation types in the Simulator controller. One such example would be an interaction between an object simulated using FEM and another using the PhysX. We also plan to expand the supported simulation types in the Simulator controller with dynamic FEM, and nonlinear simulation using a novel data-driven approach known as PhyNNeSS [8]. We will also add more collision detection methods to our support library. GPU-based computing has the potential to reduce computational load on CPU and increase computational speed. Currently, GPU-based computation is indirectly supported in our framework through PhysX. In the future, we plan to implement support for GPU-based computation directly in our framework.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

The authors gratefully acknowledge the support of NIH/NIBIB through grant #R01EB005807 and CNPq-Brazil through grants #481268/2008-1 and #302679/2009-0. The authors also thank Dr. Blake Hannaford and Hawkeye King from the University of Washington for hosting our packet reflector.

References

1. Adl-Tabatabai, A-R.; Gross, T.; Lueh, G-Y. Code reuse in an optimizing compiler. OOPSLA' 96. Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM; 1996. p. 51-68.
2. Allard, J.; Cotin, S.; Faure, F.; Bensoussan, P-J.; Poyer, F.; Duriez, C.; Delingette, H.; Grisoni, L. Medicine meets virtual reality (MMVR'15). 2007. SOFA an open source framework for medical simulation.
3. Ayache, N.; Delingette, H. Surgery simulation and soft tissue modeling, international symposium, IS4TM 2003; Juan-Les-Pins, France. June 12–13; Springer; 2003.
4. Basdogan C, De S, Kim J, Muniyandi M, Kim H, Srinivasan MA. Haptics in minimally invasive surgical simulation and training. IEEE Comput Graph Appl. 2004; 24(2):56–64. [PubMed: 15387229]
5. 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'; New York, NY, USA: ACM; 2007. p. 281-288.
6. Buttolo P, Oboe R, Hannaford B. Architectures for shared haptic virtual environments. Compt Graph. 2000; 21(4):478–483.
7. 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. Presence: teleoperators and virtual environments, 1997. 2006; 6:147–159.
8. Deo D, De S. A physics-driven neural networks-based surgery simulation system (PhyNNeSS) for multimodal interactive virtual environments involving nonlinear deformation objects. Submitted to Presence.
9. El-Far NR, Nourian S, Zhou Jilin, Hamam A, Shen Xiaojun, Georganas ND. A cataract tele-surgery training application in a hapto-visual collaborative environment running over the CANARIE photonic network, Haptic audio visual environments and their applications. IEEE Int Workshop. 2005; 4:1–2.
10. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional; Massachusetts: 1995.
11. Gottschalk, S.; Lin, MC.; Manocha, D. OBBTree: a hierarchical structure for rapid interference detection. SIGGRAPH'96. Proceedings of the 23rd annual conference on computer graphics and interactive techniques; ACM Press; 1996. p. 171-180.
12. Gunn, C.; Hutchins, M.; Adcock, M.; Hawkins, R. Stud health technol inform 98. IOS Press; 2004. Surgical training using haptics over long internet distances; p. 121-123.
13. Hermann, E.; Raffin, B.; Faure, F. Interactive Physical Simulation on Multicore Architectures. Eurographics workshop on parallel and graphics and visualization, EGPGV'09; Munich, Germany. March 2009; 2009.
14. Jeffrey H, Gerald MF, Lee LS, Nathaniel JS, Lelan FS, Bruce S, Kaaren H. The SAGES FLS Committee. Development and validation of a comprehensive program of education and assessment of the basic fundamentals of laparoscopic surgery. Surgery. 2004; 135(1):21–27. [PubMed: 14694297]
15. Lin, MC.; Manocha, D. Fast proximity queries with swept sphere volume. Proceedings of international conference on robotics and automation; April 2000; 2000. p. 3719-3726.
16. Liu A, Tendick F, Cleary K, Kaufmann C. A survey of surgical simulation: applications, technology, and education. Presence. 2003; 12(6):599–614.
17. Teschner M, Kimmerle S, Heidelberger B, Zachmann G, Raghupathi L, Fuhrmann A, Cani M-P, Faure F, Magnenat-Thalmann N, Strasser W, Volino P. Collision detection for deformable objects. Comput Graph Forum. 2005; 24(1):61–81.
18. Maciel A, Halic T, Lu Z, Nedel LP, De S. Using the PhysX engine for physics-based virtual surgery with force feedback. Int J Med Robot. 2009; 5:341–353. [PubMed: 19449317]
19. Maciel A, De S. An efficient dynamic point algorithm for line-based collision detection in real time virtual environments involving haptics. Comput Animat Virtual Worlds. 2008; 19(2):151–163.

20. Maciel A, Liu Y, Ahn W, Singh TP, Dunnican W, De S. Development of the VBLaST™: a virtual basic laparoscopic skill trainer. Int J Med Robot. 2008; 4(2):131–138. [PubMed: 18348181]
21. Maciel A, Boulic R, Thalmann D. Efficient collision detection within deforming spherical sliding contact. IEEE transactions on visualization and computer graphics. IEEE Comput Soci. 2007; 13:518–529.
22. 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; New York: ACM; 2007. p. 273-280.
23. Ponder, M.; Papagiannakis, G.; Molet, T.; Magnenat-Thalmann, N.; Thalmann, D. VHD++ development framework: towards extendible, component based VR/AR simulation engine featuring advanced virtual character technologies. Proceedings computer graphics international; 2003; IEEE Computer Society; 2003.
24. Pree, W. Meta patterns—A means for capturing the essentials of reusable object-oriented design, ECOOP'94. Proceedings of the 8th European conference on object-oriented programming; Springer; 1994. p. 150-162.(frozen spots/hot spots)
25. Rihele, D. Framework design: a role modeling approach. ETH Zürich; Zürich: 2000.
26. Ruffaldi, E.; Frisoli, A.; Bergamasco, M.; Gottlieb, C.; Tecchia, F. A haptic toolkit for the development of immersive and web-enabled games. VRST'06: Proceedings of the ACM symposium on virtual reality software and technology; New York: ACM; 2006. p. 320-323.
27. Saupin, G.; Duriez, C.; Cotin, S. Contact model for haptic medical simulations. ISBMS; Switzerland: 2008. p. 157-165.
28. Schroeder, W.; Martin, K.; Lorensen, B. The visualization tool-kit: an object-oriented approach to 3-D graphics. Prentice Hall; New Jersey: 1997.
29. Seugling, A.; Rölin, M. Evaluation of physics engines and implementation of a physics module in a 3d-authoring tool. Umea University; Sweden: 2006.
30. Shaw, C.; Green, M.; Liang, J.; Sun, Y. ACM transactions information system. Vol. 11. ACM; New York: 1993. Decoupled simulation in virtual reality with the MR toolkit; p. 287-317.
31. Vlissides JM, Linton MA. Unidraw: a framework for building domain-specific graphical editors. ACM Trans Inf Syst. 1990; 8:237–268.
32. Yeongho K, Jeong-Ho C, Jongki L, Myeng KK, Nam KK, Jin SY, Yong OK. Collaborative surgical simulation over the internet. Int Comput IEEE. 2001; 5(3):65–73.
33. Zhou, J.; Shen, X.; Georganas, ND. Haptic tele-surgery simulation, haptic, audio and visual environments and their applications. HAVE, 2004. Proceedings the 3rd IEEE international workshop on; 2–3, October 2004; 2004. p. 99-104.
34. Zhou, J.; Shen, X.; Saddik, AE.; Georganas, ND. Design and implementation of haptic telementoring over the internet. Proceedings of the 11th IEEE international symposium on distributed simulation and real-time applications; October 22–26, 2007; 2007.
35. Zhu, B.; Gu, L.; Peng, X.; Zhou, Z. A point-based simulation framework for minimally invasive surgery. In: Székely, et al., editors. Biomedical simulation; 5th international symposium, ISBMS 2010; Springer, LNCS; 2010.

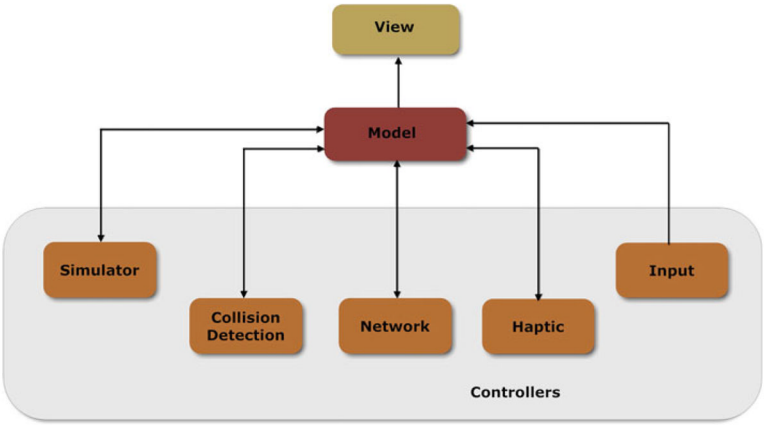

Fig. 1.
The structure of the extended model-view-controller (MVC) framework

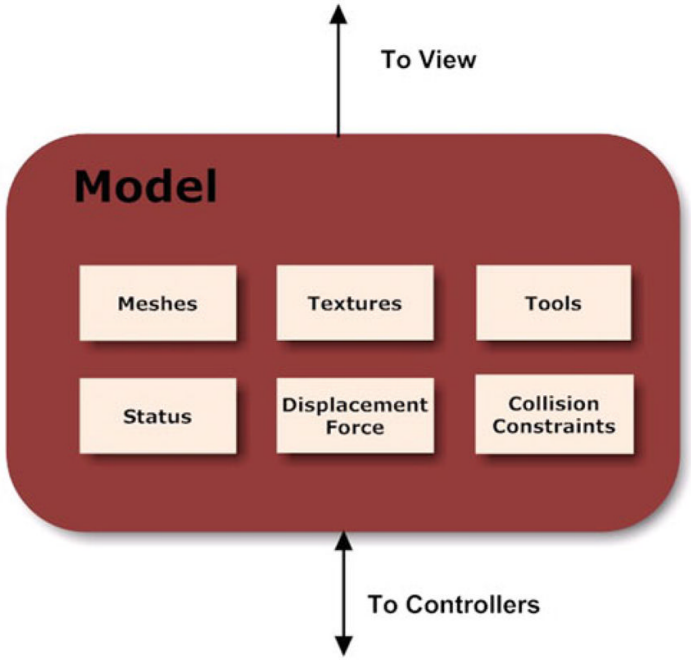

Fig. 2.
The structure of the Model module

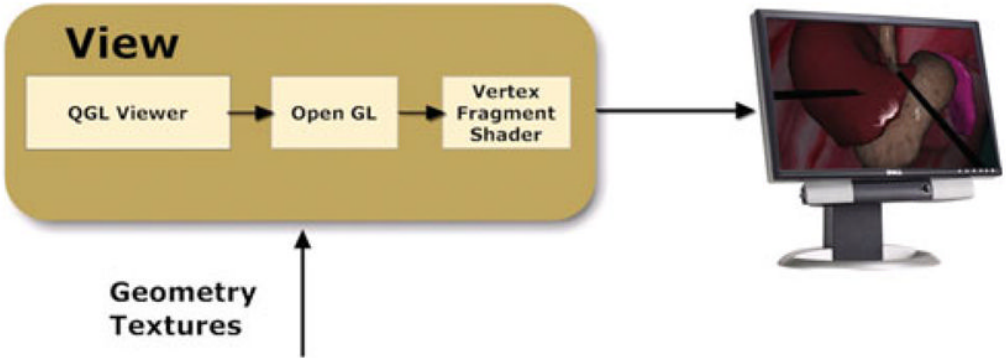

Fig. 3.
The structure of the View module

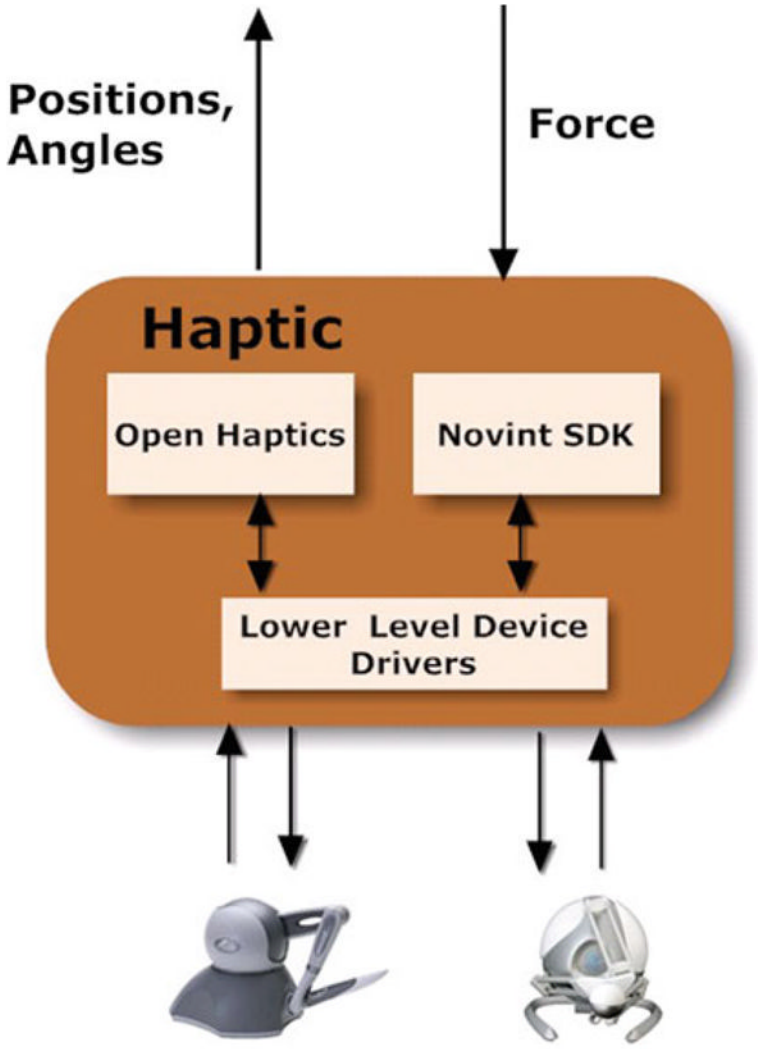

Fig. 4.
The structure of the Haptics controller

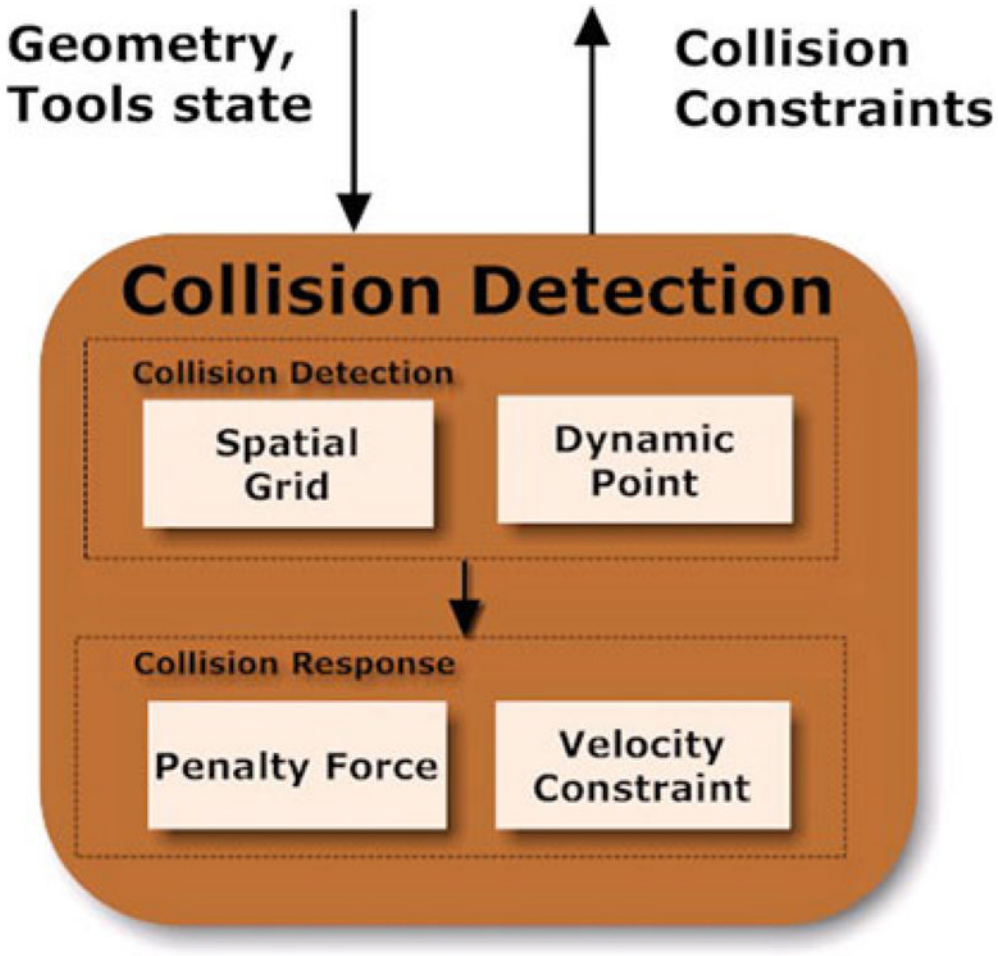

Fig. 5.
The structure of the Collision Detection controller

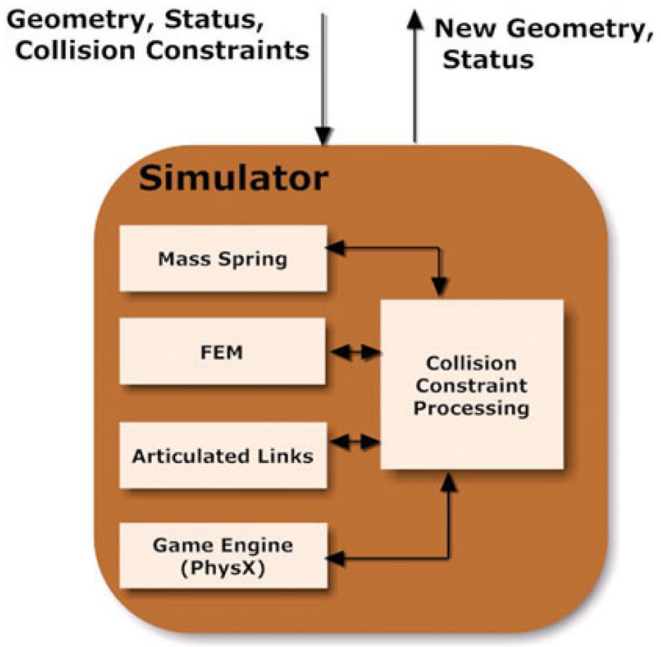

Fig. 6.
The structure of the Simulator controller

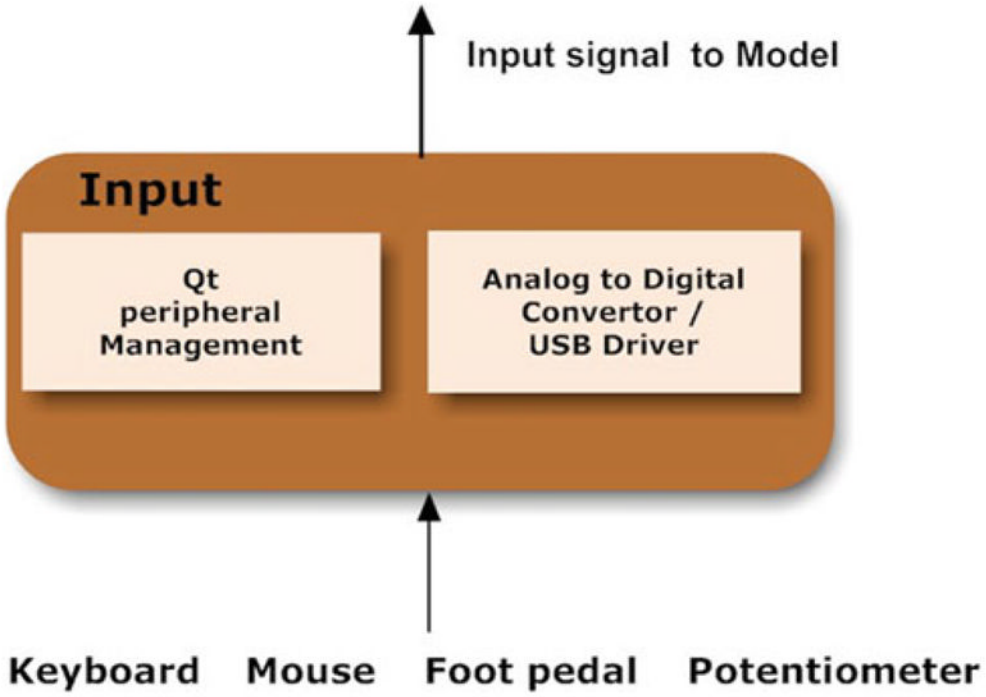

Fig. 7.
The structure of the Input controller

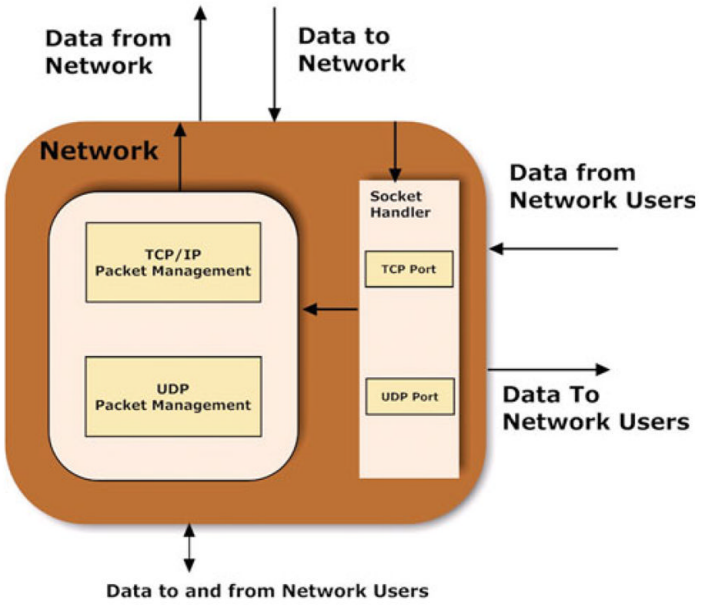

Fig. 8.
The structure of the Network controller

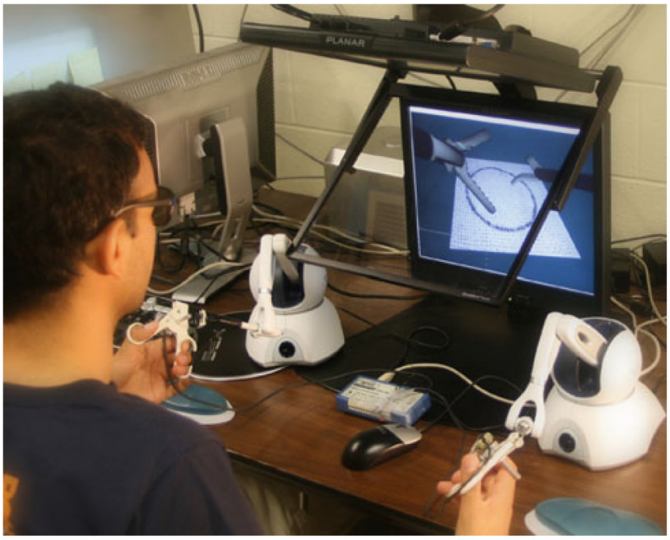

Fig. 9.
The hardware setup of the simulator comprises a stereo monitor with polarized glasses and two SensAble Phantom Omnis with custom surgical handles

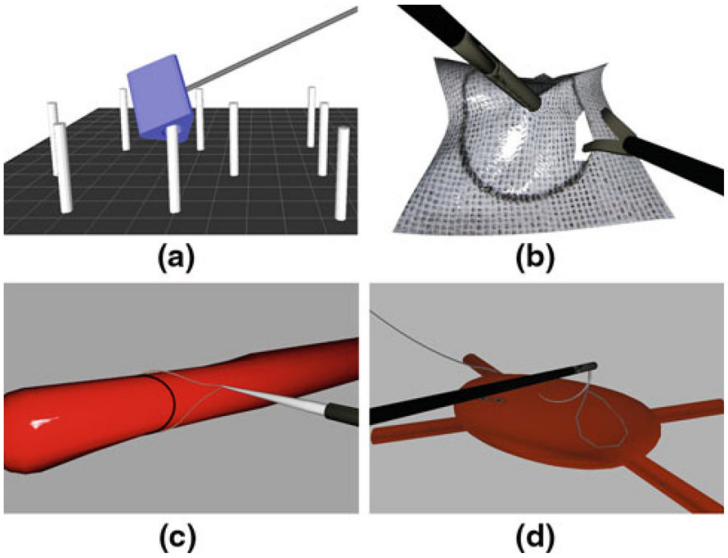

Fig. 10.
The 4 VBLaST tasks: **a** peg transfer; **b** circular pattern cutting; **c** ligating loop; **d** suturing

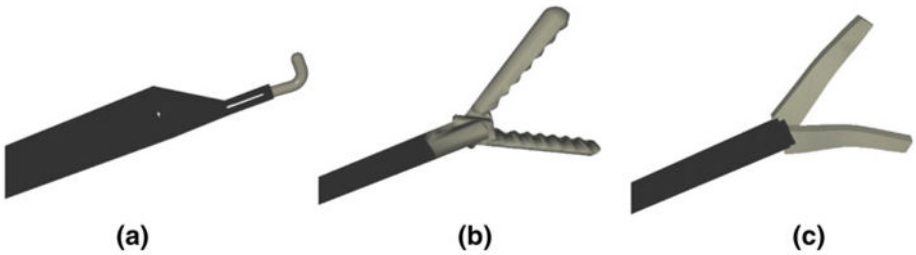

Fig. 11.
Models of the laparoscopic instruments: **a** hook cautery, **b** grasper and **c** scissors

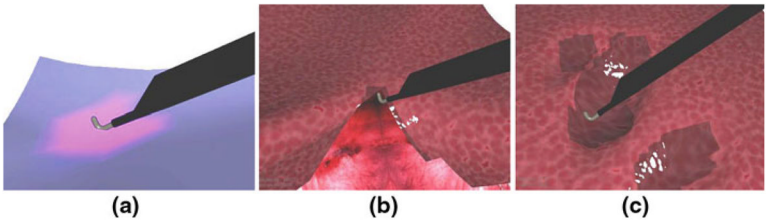

Fig. 12.

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 a more massive organ

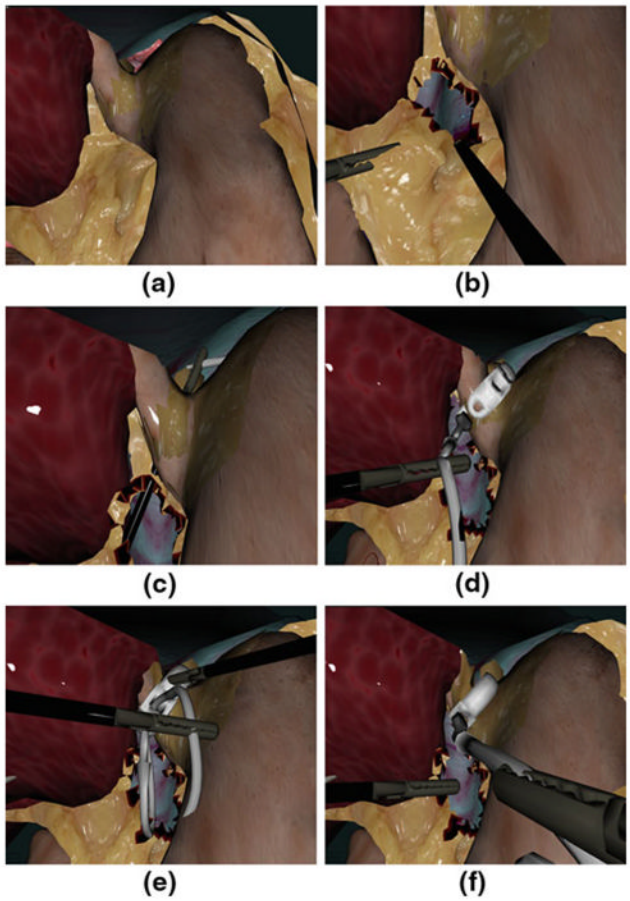

Fig. 13.

Snapshot from LAGB simulation illustrating the various steps in the band placement: **a** liver retracted to expose lesser omentum; **b** dissection of the lesser omentum; **c** exposure of angle of his and picking of the band tip; **d** band placement on the stomach; **e** sliding of the band through the hole for locking; **f** band locked and secured in place

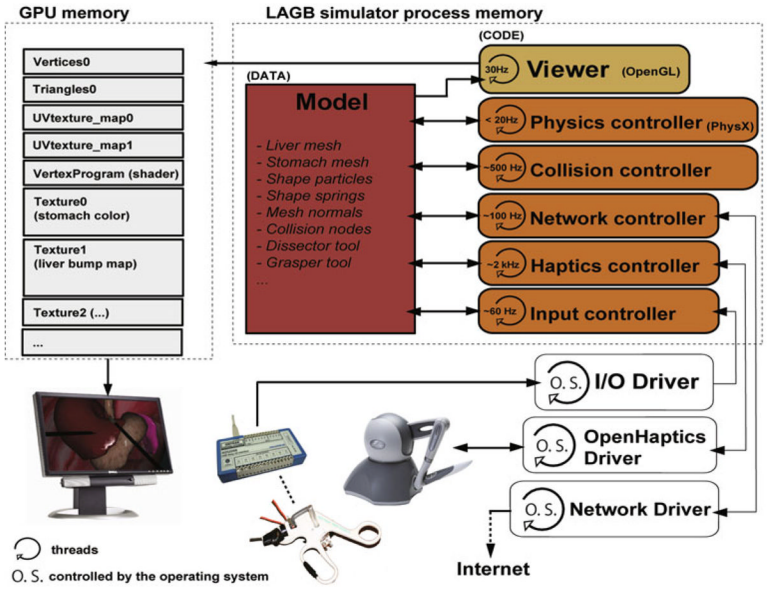

Fig. 14.

Status of a system running the LAGB application within our framework. The schematic shows the simulation running with the several threads at different frequencies and how they communicate through the model and with the external world. Color schema has been preserved for coherence with Figs. 1–8

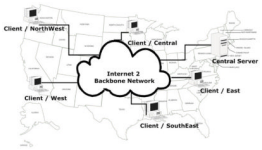

Fig. 15.

Example of a Networked Haptic Surgical Environment with a massively parallel high performance computing platform simulating complex surgical scenarios and multiple clients across USA connected to the simulation

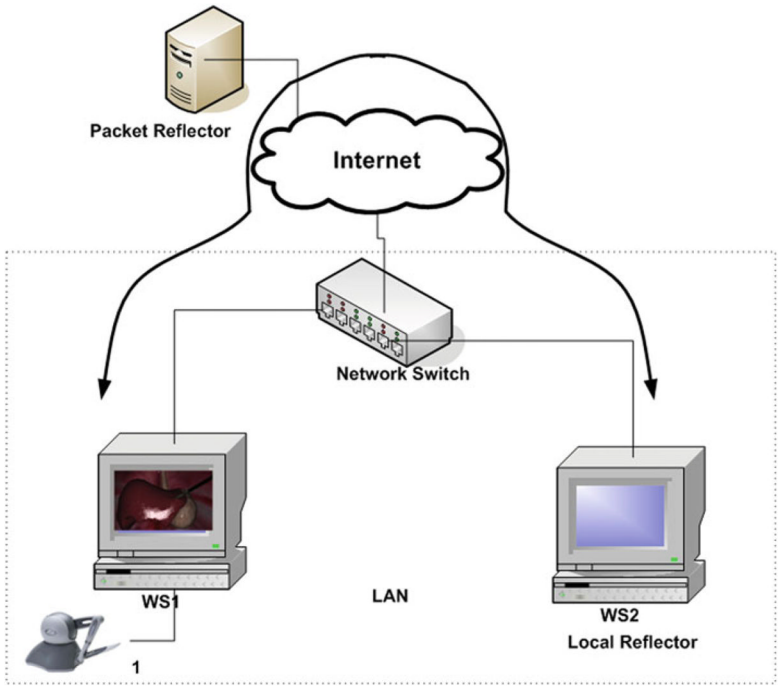

Fig. 16.
Tele-surgery experiment setup illustrating the packet reflector and local LAN configurations

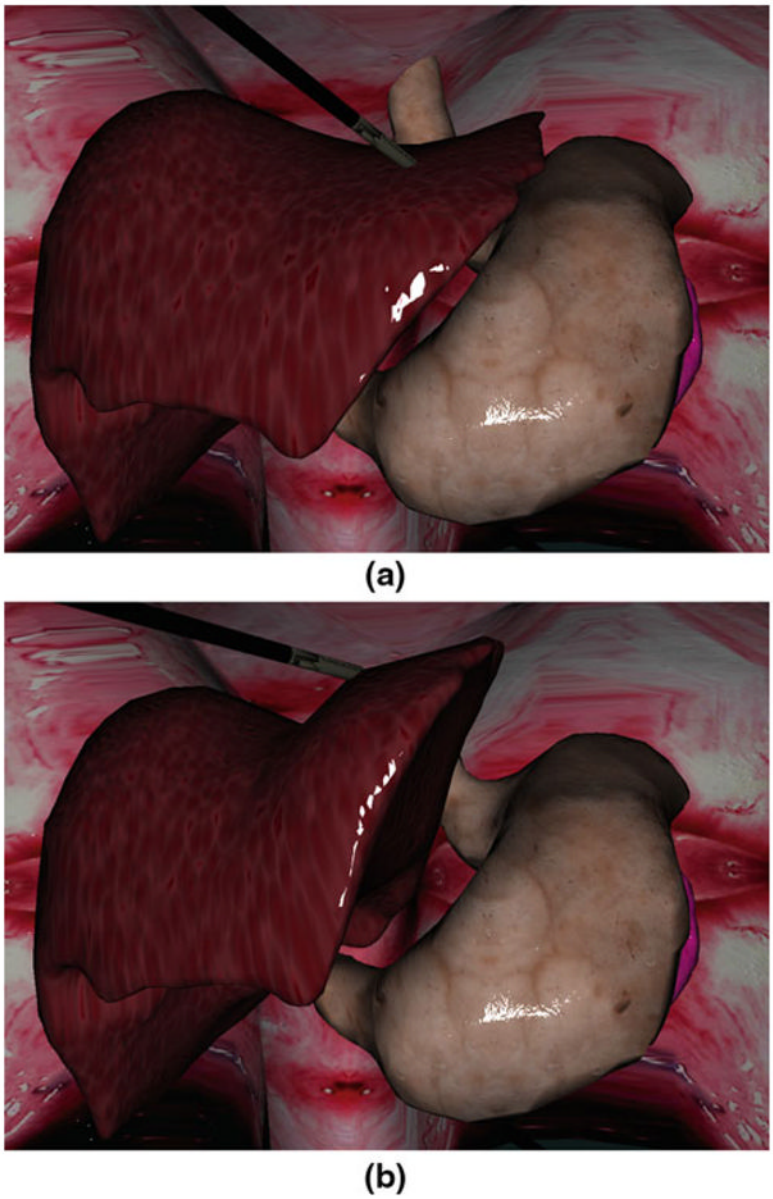

Fig. 17.
Snapshot from the tele-surgery simulation showing the remote slave interacting with the liver model: **a** making contact; **b** retracting to expose the stomach

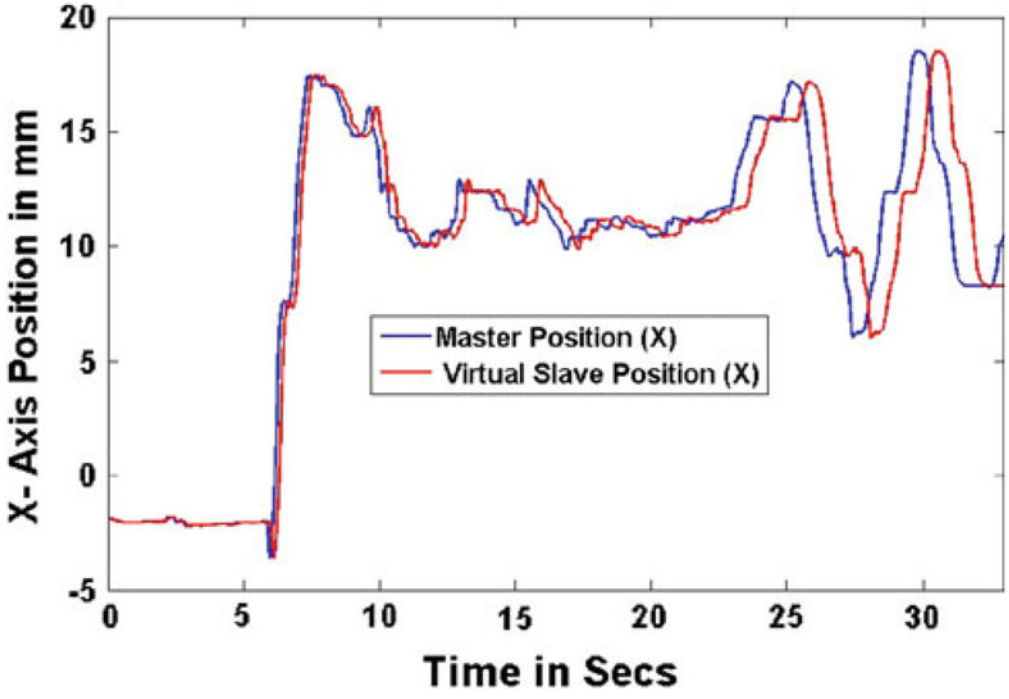

Fig. 18.

X-axis positions of master and the virtual slave devices showing tracking under latency during interactions with the deformable model

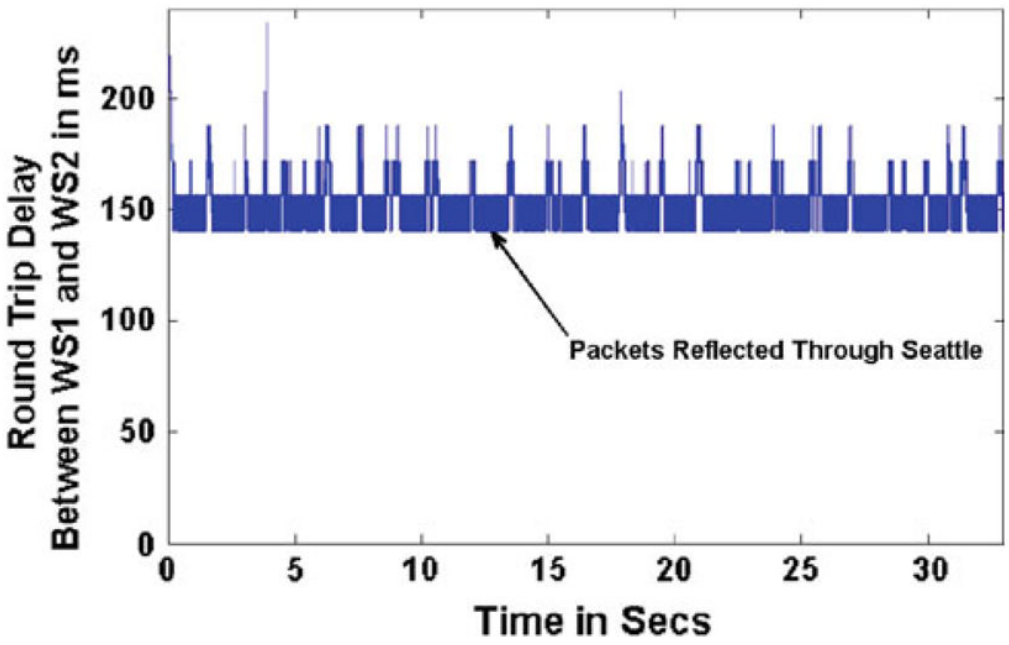

Fig. 19.
Round trip delay between WS1 and WS2 when the packets are reflected through the server in Seattle