Haptic Rendering in Interactive Applications Developed with Commodity Physics Engine

Kup-Sze Choi, Leon Sze-Ho Chan
School of Nursing, The Hong Kong Polytechnic University, Hong Kong, China
Email: kschoi@ieee.org

Jing Qin
Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, China
Email: harryqinjingcn@gmail.com

Wai-Man Pang
Computer Arts Lab, The University of Aizu, Japan
Email: wmpang@u-aizu.ac.jp

Abstract—Availability of commodity physics engines such as PhysX’s nVidia has significantly reduced the effort required for developing interactive applications concerning the simulation of the physical world. However, it becomes a problem when force feedback is needed since the addition of haptic rendering into these applications is non-trivial. The issues include the high haptic update rate and the inaccessibility of force data in the physics engine. In the paper, we tackle the first issue by mediating the update-rate disparity between haptic rendering and other processes by data buffering, and the second issue by calculating the force feedback indirectly using the engine’s collision geometry data. The major benefit of these techniques is that they enable a homogeneous development environment where the same engine can be used for both the physics and haptic simulation. Furthermore, integration of force feedback into physics-engine based applications would not introduce significant changes to developer’s codebase. The proposed techniques have potential to streamline the development of demanding applications such as virtual surgical simulation and immersive computer gaming.

Index Terms—middleware, physics engine, force feedback, haptic rendering, collision response, virtual reality

I. INTRODUCTION

Being a major human perception channel, haptic feedback plays an increasingly important role in interactive computer applications to enhance the level of realism and immersion of virtual environments. Computer-simulated feedback forces enable users to feel the virtual worlds with their sense of touch, providing a kind of interactive experience that applications with visual and audio feedback only cannot offer [1]. Among various haptic-enabled interactive applications, virtual surgical training is the one where haptic feedback is of critical importance [2, 3]. To simulate surgical operations, it is necessary to generate the feedback forces caused by tissue-tool interactions, so that trainees are able to feel the virtual tissues through their hands in a way similar to that in real surgery. In virtual surgery, realistic simulation of the physics of the objects involved is a core component of surgical simulator, based on which timely and accurate visual and haptic feedback of the operative procedures are rendered interactively for the users. The physical simulation is a difficult and laborious task however. Incidentally, commodity physics engines have been employed to facilitate this task [4, 5], although they are primarily developed for computer gaming and graphics rendering. From a developer’s point of view, the ability to easily couple the physics and the feedback components is a major consideration that affects the quality of the end result as well as the development cycle. However, coupling these two components is non-trivial. For example, computational speed of the physics engine may not cope with the high refresh rate required for realistic haptic rendering. Accessibility to key parameters in the source level, e.g. critical force data, of physics engine is also limited. Haptic rendering with physics engine is thus a tricky task, hindering the software development process. To address these issues, a simulation platform is presented in the paper by using a commodity physics engine, nVidia’s PhysX, and the haptic devices by Sensable Technology Inc. as the 3D user interface. Methods to integrate the haptic devices with PhysX, with emphasis on minimal development work will be discussed. The motivation of this work is to explore the potential of using commodity physics engine for haptic-enabled virtual reality applications, so as to come up with an effective approach for rapid prototype development.

The rest of the paper is organized as follows. Section II gives an overview of the physics engine PhysX with its applications in medicine and health care, and provides background information about haptic rendering and the medical applications. The commonly used haptic devices, Sensable’s Phantom, and the programming approaches are also described. Section III presents a platform proposed to enable haptic rendering for PhysX.
II. RELATED WORK

In this section, we will first focus our discussion specifically on PhysX and its applications in health care, followed by the software development of haptic-enabled applications.

A. The Physics Engine - PhysX

PhysX is a middleware by the nVidia Corporation that provides real-time physics simulation. It is available on multiple target platforms and can be hardware-accelerated when appropriate hardware, including physics processing unit (PPU) and graphics processing unit (GPU), is installed on the target machine. Most notable features of PhysX include collision detection and physics-based simulation of rigid bodies, cloth, soft bodies, and fluid [6].

There is an increasing adoption of PhysX by both the game industry and research groups, because of the associated reduction in development time for physics simulations and the well-established communities for developer support. A number of game titles has used the hardware-accelerated engine to achieve many appreciated effects like explosion and turbulence, which are smoother and faster than ever [6].

For more serious applications, Pang et al. [7] tried to accelerate physical simulations with PPU for medical training and developed an orthopedics surgical simulator. The use of PPU significantly improves the performance of soft tissue and bleeding simulation. Later, Ma et al. [8] exploited the PhysX engine for physics-enriched virtual reality (VR) rehabilitation, in which they investigated the impact of physics simulation on motor rehabilitation therapies. Recently, Maciel et al. [4] proposed solutions for constructing multimodal surgical simulation environments based on the PhysX engine. It tackled the difference in update rates between different modules in the system by introducing the model-view-controller (MVC) framework so that multiple threads can be executed in parallel at different speeds on different cores on multi-core CPU. The idea of introducing a collision handling layer was also proposed in their paper but the details were not provided. The potential of electrocautery simulation using PhysX was explored by Lu et al. [9]. They proposed an ad-hoc and decoupled method to perform haptic rendering, which was not comprehensive and generic for most VR applications.

B. Haptic Rendering in VR Applications

To render immersive experience in virtual environments, many recently developed VR systems have integrated kinesthetic and tactile feedback through the use of haptic devices. Research has demonstrated that integrating haptic sensation can greatly enhance the effectiveness of the VR based simulation system [3, 10].

While many systems have been developed with haptic sensation, most of them can only support one haptic device. In reality, manual tasks are often performed by two hands. For example, in ultrasound-guided biopsy training system, users need to manipulate an ultrasound transducer and a biopsy needle collaboratively to insert the needle into an accurate position [11, 12]. To simulate these scenarios, a pair of Phantom Omni haptic devices manufactured by the Sensable Technologies Inc., which is very popular for rendering forces in VR applications, is utilized to enable two-handed operations in the proposed simulation platform. The stylus of the device is to mimic the handle of surgical tool. Virtual objects are manipulated interactively by maneuvering the stylus. Each of these devices has 6 degrees of freedom in position/orientation input, and 3 degrees of freedom in force feedback output.

To program the Phantom haptic device, the application programming interface (API) OpenHaptics is used to make it easy and fast to develop new haptic applications or to add haptics into existing applications [13]. Within the API, there are two implementations for reading the

![Figure 1. The simulation platform.](image-url)
current position of the haptic interface and rendering feedback forces – Haptic Device API (HDAPI) and Haptic Library API (HLAPI). The HDAPI permits direct communication with haptic device to obtain haptic interface’s position and render the forces. With good understanding about the theory of haptics, developers need to formulate the haptic interactions and design the associated force equations, based on which forces are calculated to drive the haptic device directly. The developers should also ensure that the force computations are fast enough for haptic rendering (with the aid of efficient data structure and collision detection schemes) and are executed in a thread-safe process. Conversely, the HLAPI is more easy to use. It is a high-level API built on top of HDAPI. Developers only need to set the material properties of the interacting objects and the force computations for haptic rendering are handled in HLAPI. It reuses OpenGL graphics rendering code to construct a scene graph of virtual objects and a haptic rendering engine to automatically update the position and generate the feedback forces within the scene [13]. Event-driven programming is also possible with HLAPI.

In principle, HLAPI is more suitable for extending existing non-haptic systems to become haptically supported [14]. Developers can assign haptic material properties (stiffness and friction coefficients) to the geometric primitives of virtual objects of an existing system. When virtual instruments and virtual object collide, the haptic rendering engine makes use of the specified material properties along with the position data read from the haptic device to calculate the appropriate forces and send them to the haptic device. Recently, a more high-level interface has been developed based on HLAPI to further facilitate the development of systems with haptic sensation [15].

Although OpenHaptics is a widely used API for haptic rendering, VR applications making use of commodity physics engine for feedback force calculation and OpenHaptics for force rendering are not very common. Examples include the orthopedic surgery simulator developed by Qin et al. [5] and the laparoscopic surgery simulator by Maciel et al. [4]. In the first example, since the haptic interactions primarily involve the contact between the tip of the virtual blood sealer and the wound surface (point-plane contacts), haptic rendering can be achieved simply by employing HLAPI. Material properties are assigned to the geometric primitives (i.e., triangles and edges) of virtual organs, and the resulting force feedback is handled and computed with the API. The interactions in the second example are more complicated, involving several tools in laparoscopy, e.g., hook cautery, grasper and scissors, and various virtual tissues. While it is reported that PhysX is utilized for the detection of tool-tissue collisions, details regarding the haptic model, force equations or the API employed for haptic rendering is not provided. In this paper, we use HDAPI for haptic rendering because of its flexibility in producing various force effects, and its independence of graphics APIs (such as DirectX) employed for application design. This makes it more convenient for our platform to adopt different force models or customized approaches to compute the forces for haptic rendering. Our goal is to couple HDAPI with the PhysX engine in order to provide users with an effective and flexible platform for developing haptic-enabled interactive applications, especially those involving two-handed operations.

### III. HAPTIC-ENABLING PLATFORM FOR PHYSX

A problem of using PhysX with HDAPI is that the force data associated with the dynamics of objects, which are needed to compute the feedback force for haptic rendering, are not accessible by developers. Even when the force data are available under some conditions, inaccuracy in the solver produces noticeable jitters in the feedback force. The Force Simulation Layer (FSL) is proposed to alleviate the above problems by calculating the feedback forces with the collision geometric data provided by PhysX, and thereby rendering the forces for the haptic devices via HDAPI. While FSL serves the same purpose as that of the MVC framework [4], it is based on data buffering and does not require a multi-core CPU as in the latter approach.

The architecture of the simulation platform is shown in Fig. 1. It does not significantly differ from a typical PhysX application except with the addition of haptic devices and the implementation of the Force Simulation Layer between PhysX and HDAPI. The FSL acquires the interface position of the currently selected haptic device and transforms this position into the rendering scene coordinate. The object in the scene, as attached to a haptic interface, gets updated with its new position. PhysX uses the new position of the object and advances the physics simulation of the entire scene. Collisions detected by PhysX are reported back to the FSL to compute appropriate resultant force, and to drive the haptic device via the HDAPI. The sequence of data flow between PhysX, FSL and HDAPI is shown in Fig. 2.
Since this particular PhysX simulation runs at 60 Hz whereas the haptic interface is required to run at 1 kHz asynchronously, the FSL provides two memory buffers for each haptic device, one for storing haptic interface position and the other for storing collision and dynamics data, as depicted in Fig. 3. These buffers can be safely accessed and modified asynchronously. The processing routine that computes and updates the feedback force runs synchronously with the haptic interface at 1 kHz. All haptic devices are synchronously updated in the same processing routine. On the haptic side, the position buffer is refreshed during each HDAPI update loop. The position can be read by calling the hdGetDoublev function with either HD_CURRENT_POSITION or HD_CURRENT_TRANSFORM being set. Similarly, a force can be sent to the current haptic device by calling hdSetDoublev and specifying HD_CURRENT_FORCE.

On the PhysX side, collisions are intercepted by calling the NxUserContactModify::onContactConstraint function. Whenever a collision is detected on an object and the contact modification feature is enabled, this function will be called to extract useful geometric information about the collision, such as the penetration, contact point and contact normal, for force feedback computation.

IV. FEEDBACK FORCE COMPUTATION

On top of the FSL, three methods for producing collision responses are devised to calculate feedback forces by using the collision geometry data obtained from PhysX. Depending on the chosen method, the structure of the processing routine and the data received from PhysX would vary.

A. Penetration of Colliding Object Pair

With the PhysX engine, when a collision between a pair of objects occurs, the collision geometry data become available for the application. Specifically, PhysX automatically computes the penetration depth \( e \) (called as \( \text{error} \) in PhysX), contact point \( \mathbf{p}_c \), and unit contact normal \( \mathbf{n} \) of the two colliding objects as shown in Fig. 4. If the objects are considered to have a combined material stiffness \( k \), then the reaction force \( F \) can be modeled by the Hooke’s Law as

\[
F = ke. \tag{1}
\]

Since the penetration depth is along the direction of the unit contact normal,\[ F = ke \mathbf{n}. \tag{2}\]

Here, we assume that the material is linearly elastic such that the reaction force is modeled as being linearly dependent on the penetration depth – the deeper the penetration the stronger the reaction force. The equations above only model the hardness of the object using the value of \( k \). An attempt to pierce into an object modeled with a large \( k \) would produce a stronger reaction force that makes the penetration more difficult than that modeled with a small \( k \).

The penetration depth is updated in every PhysX simulation step at 60 Hz; however, the haptic device requires a refresh rate of 1 kHz. Therefore, between PhysX simulation steps, a new feedback force \( \mathbf{F}' \) can be extrapolated by using the current haptic position \( \mathbf{p}_i \) and the haptic position in the most recent simulation step \( \mathbf{p}_{i-1} \) as follows,

\[
\Delta \mathbf{p}_n = (\mathbf{p}_i - \mathbf{p}_{i-1}) \cdot \mathbf{n} \tag{3}
\]

Figure 4. Collision geometry data provided by PhysX include penetration depth \( e \), contact point \( \mathbf{p}_c \), and contact unit normal vector \( \mathbf{n} \). Contact plane \( P \) can be calculated using \( \mathbf{p}_c \) and \( \mathbf{n} \).
\[ F = k(\Delta p_n + e)n \]  

The calculation described above approximates the feedback force during each haptic update. The function NxUserContactModify::onContactConstraint executes in each PhysX simulation step as long as two objects are in contact with each other. Before the next PhysX update, the penetration depth \( e \) remains the same in the equation because \( p_{i+1} \) is constant between two PhysX simulation updates.

This method is experimented using a virtual horizontal plane positioned at \( y = 3 \) and a virtual object to be moved by the haptic device on the plane along the \( x \)- and \( z \)-axis respectively. Note that in this experiment, the setting of stiffness \( k \) is not based on the true physics of the material. It is set manually such that the computed force is within a range that does not exceed the maximum force limit of the haptic device. This is to prevent from damaging the device. The material stiffness is then set to \( k = 1.0 \). The variations in haptic interface positions for motions along the \( x \)- and \( z \)-axis are shown in Fig. 5. Surprisingly, it is found that the motion along the \( z \)-axis has a less stable output than that along the \( x \)-axis. The \( z \)-axis motion produces plenty of jitters on the haptic device. Attempts have been made to identify the problem by using other sets of haptic devices but the situations are similar. It is suspected that PhysX has different solver accuracy along different axes when performing the simulations.

\section*{B. Penetration through Contact Plane}

Instead of using the penetration depth, an alternative method to compute the feedback force is to make use of the other PhysX’s collision geometry data, i.e. contact point and contact normal. Here, a contact plane is constructed from the contact point and contact normal. The haptic interface is then restricted to move on one side of the contact plane. Any attempt to penetrate through the plane will be resisted by a force according to the penetration depth and the material stiffness, modeled by the same way as described in the last section.

Refer to Fig. 4, the contact plane \( P \) can be obtained from the contact normal \( n \) and contact point \( p_c \):
\[ Ax + By + Cz = D, \]  

where \( A, B, \) and \( C \) correspond to the \( x \)-, \( y \)-, and \( z \)-component of \( n \) as given by PhysX. \( D \) can be calculated by substituting the contact point into the plane equation to solve for \( D \). It can be created readily with PhysX’s plane creation routine NxPlane.

Once the contact plane is defined, the haptic interface can be tested against this plane for collision and the penetration during each haptic update can be calculated. The penetration depth is obtained by finding the distance \( e \) from the haptic interface position \( p(p_x, p_y, p_z) \) to the contact plane, that is,
\[ e = \frac{Ap_x + Bp_y + Cp_z - D}{\sqrt{A^2 + B^2 + C^2}}, \]  

which can be obtained by calling the NxPlane::distance function in PhysX. Whenever the penetration depth

Figure 5. Motion jittering along the \( z \)- and \( x \)-axis when penetration of colliding object pair is used for force computation.
becomes negative in the direction of the contact normal, the feedback force is updated using the Hooke's Law. Using the same test conditions as that used for evaluating the first method, the variation in haptic interface position variations are measured and shown in Fig. 6. The issue of axis-dependent motion jittering discussed previously also exists when the contact plane is used for generating collision response. However, when compared with the case where collision response is determined by the penetration depth supplied by PhysX, using the contact plane as obtained indirectly with PhysX’s contact point and normal for force computation is able to produce more stable motion and less jittering.

C. Pulling Force Simulation

The methods above are suitable for simulating the feedback forces when haptic device is used to push against an object. For the generation of feedback forces when haptic device is used to pull an object, a different approach is needed to compute the collision response. From classical mechanics, a system can be isolated with equal but opposite forces. By isolating the haptic interface from the rest of the system, the internal force becomes exposed as illustrated in Fig. 7.

Assuming that the haptic interface is massless, the reaction force would be equal to the pulling force in magnitude with an opposite direction, which can be used to provide force feedback to the haptic device. Unfortunately, PhysX does not make the reaction force available to the application. A simple solution to this problem is to simulate the force indirectly using a spring introduced between the haptic interface and the rest of the system. By measuring the elongation of the spring, the reaction force can be deduced by using the Hooke's Law. In PhysX, the distance joint is provided to model an elastic spring connecting two objects. To simulate the pulling force, this joint is used to link the haptic interface and the rest of the system while acting as a spring with constant stiffness $k$.

During the simulation, the length of the spring (joint) and the positions of ends of the two ends of the springs, $p_S$ and $p_H$, are updated and available for query. The reaction force (and hence the feedback force) $F$ can be obtained using the equation below:

$$F = k_0(p_S - p_H),$$

where $k_0$ is the spring stiffness. An experiment is conducted to test this method. A massive virtual plate is created for the experiment. The user is required lift the
plate using the haptic device and the pulling force is simulated with the method described above. The setting is shown in Fig. 8(a). Similar to the previous experiments, the spring stiffness is set manually so that the calculated pulling force will not be too large and damage the haptic device. In the experiment, the user is able to feel the simulated pulling force during the lifting process. The pulling force during the process is measured and shown in Fig. 8(b). The force oscillates at the beginning and then approaches a steady state. The result suggests that the proposed method can be used to simulate the pulling forces during haptic interactions.

V. DISCUSSION

While commodity physics engines have been adopted to reduce the effort required for physics simulation, it does not have much support for haptic rendering which is an important requirement for virtual surgery. In this study, attempts have been made to enable haptic rendering for interactive applications developed with PhysX, so that these applications can still enjoy the benefit of PhysX even though force feedback is needed. This is achieved by integrating the OpenHaptics, a commonly used API for haptic rendering, with PhysX. To deal with the problem of update-rate disparity among the visualization, physics and haptic simulation processes, the FSL is proposed to buffer the haptic interface data generated by haptic device at 1 kHz, and the collision geometry and dynamics data produced by PhysX at 60 Hz.

Three force computation methods are then developed and implemented on top of the force simulation layer. In the first approach, the penetration depth between a pair of colliding objects, provided by PhysX, is used to calculate the resulting feedback force. Experiments show that simulated force fluctuates quite considerably, especially when the haptic interface point moves along the z-direction. In the second method, the contact point and normal, also provided by PhysX, is used instead to simulate collision response. These two pieces of data are used to compute the contact plane, where the haptic interface is restricted to move on this plane. The amount of penetration below this plane is then used to calculate the feedback force. This method results in more stable feedback response when compared with the first method. For both methods, the fluctuation in feedback force when the haptic interface point moves along in z-direction is greater than that along the x-direction. The finding suggests that there is a direction-dependent issue concerning the accuracy of PhysX’s solver. The third method is developed to simulate the feedback force when a virtual object is being pulled. The pulling force is obtained by connecting an elastic spring between the haptic interface and the object, which is achieved by using PhysX’s distance joint. The pulling force is then given by the elongation of the spring and the material stiffness. Experiments show that this method is feasible for haptic simulation of virtual objects involving pulling forces.

Notice in the experiments that the material properties and the forces calculated are not based on the true physics, and therefore we do not attempt to simulate the “real” or “required” reaction forces exhibited by a certain physical material. As discussed earlier, the Hooke’s law is assumed in the simulation and the stiffness $k$ is adjusted manually so that the force would not destroy the haptic device for a given range of desired penetration depth. To simulate the behavior of real material, one may employ heuristic approaches that tune the model parameters automatically until the simulated response is similar to that of the real tissue [16], or adjust the parameters manually until the response is seemingly realistic in an interactive manner.

The purpose of our work is to suggest a way to develop applications with haptic feedback by capitalizing on physics engine. Some techniques have been proposed to achieve this goal. It is however not to improve the computational performance or visual appearance of
existing systems but merely to provide a convenient means of software development for haptic-enabled applications. Experiments are thus conducted to evaluate the feasibility of these techniques in order to demonstrate that PhysX can be also used for haptic rendering although it is not primarily designed for that purpose.

VI. CONCLUSION

In demanding real-time interactive applications like virtual surgery, haptic rendering is playing an important role on par with graphics rendering and physics simulation. The use of commodity physics engines is able to facilitate physically realistic simulation and the visualization of the simulation results, but it is yet to be fully compatible the simulation of haptic feedback. While PhysX primarily deals with real-time simulation of the physical world and the visualization of the simulated results, haptic rendering is not concerned. It does not provide developers with the force data even though they are inherently involved in the physics simulation. The proposed force computation methods can therefore be applied if PhysX is to be used for building haptic-enabled applications. This paper attempts to bridge this gap by proposing a simulation platform and providing simple and direct methods for haptic rendering by coupling PhysX and OpenHaptics. These techniques have potential for reducing development time and effort, enabling developers of haptic applications to enjoy the benefits of PhysX as well. Based on the results of this study, interactive 3D virtual surgical training applications, e.g. virtual suturing, will be developed and rapid development of a prototype simulator is anticipated. Besides, computer gaming is another application domain where the proposed techniques can be integrated with a game engine [17] to enhance the gaming experience by haptic interactions.

ACKNOWLEDGMENT

This work was supported in part by the Research Grants Council of Hong Kong SAR (Project No. PolyU 5152/09E) and the Hong Kong Polytechnic University (Project Account Code 1-ZV6C, 1-ZV2U and G-U509).

REFERENCES


Kup-Sze Choi received his Ph.D. degree in computer science and engineering from the Chinese University of Hong Kong. He is currently an assistant professor at the School of Nursing, the Hong Kong Polytechnic University, and the leader of the Technology in Health Care research team. His research interests include computer graphics, virtual reality, physically based simulation, computational intelligence, and their applications in medicine and health care.

Sze-Ho Chan received his B.Sc. and M.Sc. degrees in mechanical engineering from the University of British Columbia, Canada. He was engaged in the programming tasks and software development of the project.
Jing Qin received his B.Eng. and M.Eng. degrees from the Institute of Information Engineering of the University of Science and Technology Beijing, China. He continued his graduate study and received his Ph.D. degree from the Department of Computer Science and Engineering of The Chinese University of Hong Kong in 2009. His research interest lies in the broad area of computer-assisted surgery.

Wai-Man Pang is currently an assistant professor in the Computer Arts Lab., University of Aizu, Japan. His research interests include non-photorealistic rendering, image-based rendering, GPU programming, and physically based deformation. Pang is the author of several significant journals or books including ACM Transaction on Graphics, IEEE TVCG, IEEE CG&A and ShaderX5.