Comparison of Algorithms for Haptic Interaction With Isosurfaces Extracted From Volumetric Datasets

Silvio H. Rizzi
e-mail: srizzi2@uic.edu

Cristian J. Luciano
e-mail: clucia1@uic.edu

P. Pat Banerjee[1]
e-mail: banerjee@uic.edu

2039 Engineering Research Facility,
University of Illinois at Chicago,
842 West Taylor Street, Chicago, IL 60607

Combinations of graphics and haptics libraries are used in medical simulations for simultaneous visualization and tactile interaction with complex 3D anatomy models. The minimum frame rate of 1 kHz for haptics rendering makes it a nontrivial problem when dealing with complex and highly detailed polygonal models. Multiple haptics algorithms based on polygonal mesh rendering, volume haptics, and intermediate representation are evaluated in terms of their servoloop rendering time, client thread rendering time, and quality of force feedback. Algorithms include OpenHaptics' Feedback Buffer and Depth Buffer, GodObject and Ruspini renderers in H3D, CHAI3D implementation in H3D, ScalarSurfaceFriction mode in Volume Haptics ToolKit (VHTK), and the authors' intermediate representation algorithm based on volumetric data. The latter, in combination with surface graphics visualization, is found to deliver the best rendering time, to detect all collisions and to provide correct haptic feedback where other algorithms fail. [DOI: 10.1115/1.4006465]

1 Introduction

Volumetric datasets have become essential in virtual reality and haptics simulation for medical and surgical training. Technologies such as computed tomography (CT) and magnetic resonance imaging provide 3D scans of patient anatomy, from which 3D models can be generated to represent highly detailed and complex anatomical structures such as bone, organs, or muscle.

Traditionally, 3D models have been represented as polygonal meshes, i.e., surfaces in 3D space consisting of multiple triangles. From the volumetric data, polygonal meshes are constructed using an isosurface extraction algorithm such as Marching Cubes [1]. Furthermore, mesh processing may be required in order to reduce the number of triangles in each model (decimation) and to obtain smooth surfaces. All these preprocessing stages often demand a fair amount of time to generate high quality models. Although methods to automate the segmentation and isosurface creation processes have been discussed in the literature [2], they still require considerable human intervention.

Alternatively, anatomical 3D models can be displayed using direct volume rendering, which requires less human participation and minimum data preprocessing, generally at the expense of lower graphics frame rates.

Combinations of scene graph managers [3,4] and haptics libraries [5] have been used in surgical simulations for graphics and haptics rendering of 3D models [6,7]. The goals of these simulations are simultaneous visualization of complex 3D models and tactile interaction with anatomy at interactive frame rates. While frame rates in the order of 30–60 Hz are acceptable for graphics rendering, the minimum required rate of 1 kHz for haptics rendering makes it a nontrivial problem when dealing with complex and highly detailed polygonal models. On the other hand, volume-based haptics techniques have proven themselves capable of generating force feedback from complex anatomies at interactive frame rates [8,9].

This paper presents a comparison of haptics rendering algorithms combined with polygonal graphics rendering within the framework of medical simulation. This evaluation seeks to assess the quality of the haptic feedback provided as well as identifying the best algorithm in terms of rendering time.

2 Literature Review and Related Work

In the past, different techniques have been implemented to provide force feedback with polygonal meshes, volumetric data, and intermediate representations. This section describes some approaches found in the literature.

2.1 Polygonal Mesh Haptics Rendering. Polygonal mesh methods require 3D models to be represented as rigid polyhedra obtained from the original dataset. Within these methods, an algorithm used for single-point contacts was proposed in Ref. [10]. This method used a "GodObject" to constrain the haptic interface point to the mesh surface, avoiding penetration. The tip of the haptic device was coupled to the proxy through a spring model. In each haptic frame, the force rendered was proportional to the distance between the probe and the proxy. A virtual proxy point of finite size, to avoid fall-through due to numerical gaps in polygonal meshes, was proposed in Ref. [11]. This paper also proposed HL, a haptic interface library based on a graphics library (GL) from Silicon Graphics. The proxy method and the idea of a haptic library based on OpenGL were later implemented in SensAble's OpenHaptics [5,12]. It offered two alternative haptics rendering modes: Feedback Buffer (based on OpenGL 3D polygonal primitives) and Depth Buffer (based on the OpenGL depth buffer). Feedback Buffer delivered high quality collision detection and force feedback, but the number of polygons it can handle is limited. On the other hand, Depth Buffer was relatively insensitive to the number of polygons because it is based on a 2D image drawn on the Z-buffer. However, it exhibited "noticeable (force) discontinuities when feeling shapes with deep, narrow grooves, or tunnels" [13].

In addition to point-based algorithms, there are also line-based approaches, such as given in Ref. [14]. The authors presented a ray-based method to detect collisions between the 3D polygonal objects and the haptic stylus, which has been modeled as a line segment. When a collision was detected, the distance between the collision

[1]Corresponding author.

Contributed by the Design Engineering Division of ASME for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received April 28, 2011; final manuscript received February 1, 2012; published online April 23, 2012. Assoc. Editor: Chris Geiger.

point and the tip of the stylus was computed, and the reaction force in the normal direction was made proportional to that distance using a simple spring-damper model. Static and dynamic frictional forces were also computed in the tangential direction.

As pointed out in Refs. [15] and [16], the fact that polygonal meshes were generated from isosurfaces prevented the user from dynamically modifying the model during the simulation, since it is computationally expensive to regenerate the whole mesh in real time. Having this ability is, though, an essential requirement for modeling surgical procedures where volume removal is frequently required, e.g., bone drilling.

2.2 Volume Haptics Rendering. Iwata and Noma [17] presented an approach called volume haptization to provide force feedback from volumetric datasets. For scalar data, they mapped either the voxel values to torque vectors or the gradient of voxel values to force vectors. Avila and Sobierajski [18] described a gradient method where the normal and viscosity force components at a given point depend on the material density and the gradient magnitude at that point. The disadvantage of these methods is that they can produce instabilities or undesired vibrations, especially in regions containing sharp transitions, where the gradient magnitude and direction can vary abruptly.

In terms of using volume haptics for surgical simulation, there are significant works focusing on bone surgery. Gibson [19] developed a prototype for haptic exploration of a 3D model of a human hip based on voxels. Using occupancy maps, collisions were easily detected, and haptic feedback was generated to prevent penetration of virtual models. However, this method required the generation of occupancy maps, which is essentially a form of segmentation, demanding some preprocessing work. Petersik et al. [20] presented a haptic rendering algorithm based on a multipoint collision detection approach. A ray-based algorithm was used for graphics and haptics rendering. While static objects were represented by voxels, the location of their surfaces was obtained by a ray-casting algorithm at subvoxel resolution. This approach was found to be limited in the effective stiffness of the simulations [21]. Morris et al. [21] presented a hybrid approach based on voxels for haptics rendering and on polygonal meshes for graphics rendering. As parts of the virtual bone were being removed, their algorithm modified the surface locally and then recomputed the meshes in real time, solving the stiffness problem in Ref. [20].

Volume haptics has been systematically studied in a series of publications [22–26]. In Ref. [22], a method to generate surface and viscosity haptic feedback from volumes along with simulation of material properties was presented. The method evolved in Ref. [23], where haptic primitives, such as directed force, point, line, and plane, were used as building blocks for their proxy-based method. Based on those haptic primitives, a number of haptic modes were constructed, i.e., viscosity mode, gradient force mode, vector follow mode, and surface and friction modes. The method was refined in Ref. [24], where a numeric solver to compute the final forces was described. In Ref. [25], the VHTK was presented and implemented as an extension to SenseGraphics H3D library [4]. An analytical solver, which falls back into their numerical solver when its requirements were not satisfied, was introduced in Ref. [8]. Finally, a method that contemplates time-varying volumetric data was introduced in Ref. [26].

2.3 Intermediate Representation Methods. Intermediate representation methods were first proposed in Ref. [27]. The idea consisted of representing touchable surfaces at a given point by a virtual plane tangent to the surface at that point. The collision detection loop ran independent of the servoloop and was updated at a lower rate, whereas the servoloop was updated at a higher rate required to render stiff objects. Combining intermediate representations and lower update rates allowed simplification of the collision detection problem and quicker detection of collisions between the tip of the haptic device and the virtual plane. The method, however, had a fundamental limitation. If the update rate for the virtual plane (computed in the collision detection loop) was too low, the operator perceived discontinuities as the proxy "jumps" from one plane to another. This problem was addressed in Ref. [28], where the recovery time method was presented. The method reduced the magnitude of the force immediately after a new virtual plane was computed, allowing to gradually and smoothly bring the tip of the haptic stylus to the new surface. A simple algorithm using the intermediate representation method on volumetric data was presented in Ref. [29]. The algorithm extracted virtual planes from the volumetric data without the need of precomputing isosurfaces. This algorithm, combined with a proxy-based method, allowed the generation of haptic feedback directly from the volumetric data. Similarly, Ref. [9] presented a collision detection algorithm based on determining where a line segment intersected an isosurface defined by transfer functions that depended on voxel intensities. A line segment was created from the position of the haptic interaction point in the previous haptic frame and its current position. If a collision was detected, the point where it occurred as well as the surface normal at that point were computed. With that information, the underlying haptics library was able to compute feedback forces as if it was working with polygonal models. In Ref. [30], an intermediate local representation, which used Marching Cubes to generate isosurfaces from voxel data adjacent to the haptic stylus position, was proposed. Local isosurfaces from a $7 \times 7 \times 7$ cube were passed to the servoloop in the haptics library as an intermediate representation of the local volume data.

2.4 Evaluation of Haptic Algorithms. A number of evaluation methods for haptic algorithms have been proposed in the literature. The dependency of haptic algorithms on the user's input has been pointed out in Ref. [31]. The same paper described a methodology for evaluating haptic algorithms based on recording actual forces and trajectories from a user interacting with a real object. The recordings were used as inputs to the algorithm being tested and compared with its output. Similarly, Ref. [32] presented a virtual handwriting simulator where the position of the haptic device and its forces were recorded. For validation purposes, the haptic device was coupled to a robotic arm programmed to imitate typical inputs from a human user. In this way, the forces generated by the haptic device in response to reproducible inputs were recorded and their variations analyzed.

3 Background

3.1 OpenHaptics Architecture. SensAble haptic devices require the use of proprietary software libraries in order to access their low-level functionalities. As such, the GHOST SDK [33] initially offered by SensAble has been superseded by the OpenHaptics Toolkit [5], which provides two different application programming interfaces (API): HDAPI and HLAPI [12]. The former enables low-level access to haptic devices, whereas the latter is a high-level interface conceived to mimic OpenGL commands.

OpenHaptics is designed as a multithread API that handles: (i) a client-application thread that runs at about 30–60 Hz; (ii) a servo thread that runs at 1 kHz; and (iii) a collision thread that runs at 100 Hz. The client thread performs the graphics rendering and asynchronically sends the 3D geometry and haptic materials to the servo thread, which in turn performs the collision detection and computes the forces applied by the haptic device [12]. The collision thread is only intended to be used when the program needs to perform dynamics simulation or a more sophisticated collision detection algorithm that is more time consuming and, therefore, cannot run at 1 kHz.

Two mechanisms are provided to extend OpenHaptics' functionality, namely, *custom shapes* and *custom effects* [12]. The custom shapes mode enables application programmers to create user-defined surfaces and constraints. It is the mode used to implement the low-level functionality of algorithms such as given in Ref. [9].

Table 1 Haptic algorithms studied

	Algorithm	Rendering type	API	Nomenclature
1	OpenHaptics' Feedback Buffer [12]	Polygonal mesh rendering	H3D	FB
2	OpenHaptics' Depth Buffer [12]	Polygonal mesh rendering	H3D	DB
3	VHTK's ScalarSurfaceFriction mode [25]	Volume haptics rendering	H3D	VHTK
4	Intermediate representation algorithm in Ref. [9]	Intermediate representation methods	ImmersiveTouch	IR
5	GodObject [10]	Polygonal mesh rendering	H3D	GodObject
6	Ruspini method [11]	Polygonal mesh rendering	H3D	Ruspini
7	CHAI3D [34]	Polygonal mesh rendering	H3D	CHAI3D

On the other hand, custom effects can be implemented as part of the HLAPI, with callback functions that must be defined for starting, stopping, and computing forces as part of the servoloop.

3.2 Algorithms Evaluated. Multiple algorithms for haptic interaction with isosurface models are evaluated in this work. The selection of algorithms is based on the following criteria: (i) an implementation must be available using OpenHaptics, consequently making use of SensAble haptic devices and (ii) the algorithm must provide haptic feedback from isosurfaces. Seven algorithms satisfying the requirements were identified, as listed in Table 1.

A fundamental idea in OpenHaptics HLAPI is to emulate the interface of OpenGL [12], getting the geometry to be haptically rendered from graphics primitives. In Feedback Buffer, the first algorithm evaluated, OpenHaptics captures all the geometric primitives that generate points, lines, and polygons from OpenGL. One of the limitations of this mode is that it is required a priori to tell the API the number of vertices to be rendered for buffer allocation. In case of Depth Buffer, the second algorithm, an image rendered on the OpenGL depth buffer by a haptic camera is used to simplify computations and reduce buffering requirements. Its disadvantage is that, when disabling the haptic camera view optimization provided by OpenHaptics, only parts of the geometry visible from the viewpoint used to render the shapes are touchable. Even enabling the haptic camera view optimization, "noticeable force discontinuities are felt when touching shapes with deep, narrow grooves, and tunnels" [13].

The third algorithm evaluated, ScalarSurfaceFriction in VHTK [25], uses a plane determined by the surface gradient to detect touchable surfaces from the volumetric data. Transfer functions are used to specify the strength of the surface and its friction as a function of the voxel intensities. There is an additional parameter called distinctness which is based on the magnitude of the gradient. This mode can be used from the H3D environment using X3D and PYTHON scripts.

The fourth algorithm, IR [9], while based on voxels, preserves desirable features from polygonal mesh models. Taking advantage of OpenHaptics' custom shapes, it creates isosurfaces on-the-fly from volumetric data, which OpenHaptics uses exactly as it does with polygonal meshes. This also means that models can use stiffness, damping, dynamic, and static friction parameters as well as event callback functions (touch, untouch, motion) available to polygonal models in OpenHaptics. This algorithm runs in the OpenHaptics servoloop.

GodObject [10] and Ruspini [11] renderers are implemented as part of HAPI, a low-level layer in the H3D API [4]. H3D also includes an option to render polygonal models using the CHAI3D [34] library. Due to their availability, these implementations in H3D are the ones used in this work.

3.3 Polygonal Mesh and Volumetric Methods. In modern graphics cards, graphic pipelines are optimized for polygonal mesh models. There is also a great majority of haptic algorithms based on polygonal meshes. On the other hand, a voxel-based approach for both graphics and haptics allows one to implement volume removal procedures with relative simplicity. However, graphics volume rendering techniques are slower than polygonal mesh graphics. For this reason, some researchers have opted for using voxel-based models for haptics and volume removal procedures combined with polygonal meshes of the deformed models, which are generated on-the-fly [21]. Moreover, results in Ref. [9] also suggest that the use of volume haptics with polygon-based graphics is a promising combination in terms of efficiency. The above is one of the motivations for the comparison presented in this work. All haptics algorithms evaluated in this paper, including voxel-based algorithms, are combined with polygonal mesh graphics rendering for visualization.

4 Experiments and Results

In our experiments, the running time for each servoloop frame is measured to determine if a given algorithm is able to maintain the required rate of 1 kHz. In addition, haptics quality of the algorithms is evaluated based on the continuity of forces generated when performing a specific task. Finally, the performance of all algorithms is evaluated in terms of rendering time in the client-application thread. The details of the experimental setup used, as well as the measurement techniques employed, are shown in Appendix.

4.1 Description of the Experiments. The common part to all our experiments consists of haptically exploring anatomical models (described in Appendix), maintaining contact with the smooth surface (skull shown in Fig. 1) at all times. In the interest of generating reproducible inputs to the algorithms, we considered the approaches described in Refs. [31] and [32]. Using prerecorded trajectories and injecting them into the algorithms [31] was not possible, as OpenHaptics receives its input from the

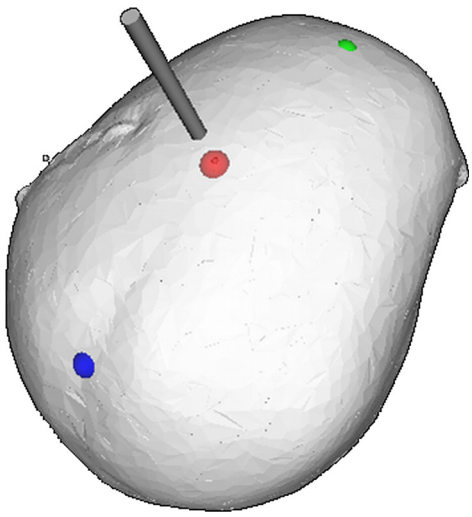

Fig. 1 Predefined trajectory for the experiments. The red sphere is animated and moves following a straight-line trajectory on the 3D surface from the green sphere (starting point) to the blue (end point) at constant velocity.

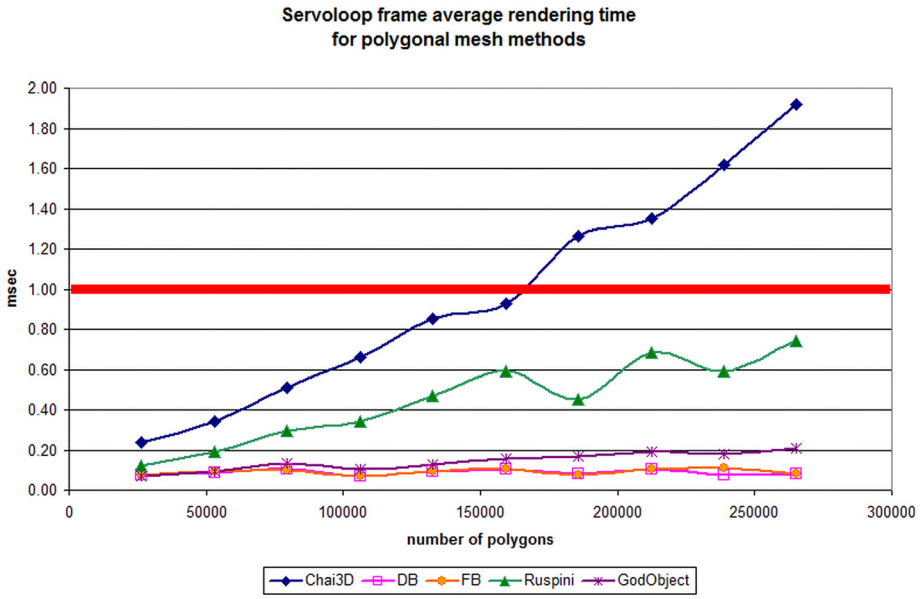

Fig. 2 Servoloop average rendering time for polygonal mesh methods

haptic device. Similarly, using another device coupled to the haptic device to provide its input [32] is not applicable, as we need to capture and evaluate the intrinsic variability of human users interacting with the models in a closed loop.

Our solution consists of having the user follow a prerecorded trajectory that has been converted to animated VRML and X3D files. In the experiments, a red sphere traverses the prerecorded path continuously and at constant speed (Fig. 1).

The operator is instructed to follow the red sphere while trying to maintain contact with the surface at all times, starting from the green sphere in Fig. 1 and ending in the blue one. This simple arrangement guarantees repeatability without the need to introduce haptic constraints to the user movement, which could affect the force rendered by the algorithms.

4.2 Experiment 1—Servoloop Frame Rate. It is well known that a minimum frame rate of 1 kHz is required to haptically display moderately stiff objects. The objective of this first experiment is assessing the performance of the algorithms by measuring their servoloop frame rate.

For haptic algorithms based on polygonal meshes, the average servoloop execution time is computed and presented in Fig. 2 as a function of the number of polygons in the model. Depth Buffer and Feedback Buffer remain insensitive to the number of polygons and their average frame rendering times are well below 1 ms. A dependence on the number of polygons is clearly visible for CHAI3D, Ruspini, and GodObject methods. The dependence is not significant for the GodObject algorithm, whereas it is more pronounced for the Ruspini and CHAI3D methods. Furthermore, for models of approximately 185K polygons and up, CHAI3D average frame rendering times are above 1 ms (represented by the bold red line in Fig. 2), which means CHAI3D is not able to maintain a haptics frame rate of 1 kHz for moderately complex polygonal models.

Volume-based haptic algorithms are independent of the number of polygons being visualized; hence, in this experiment the results for VHTK and IR are not presented as a function of the number of polygons. Instead, a normalized histogram of the sampled servoloop frame rendering time is shown in Fig. 3. For the volumetric model described in Sec. 4.1, VHTK's peak is at 0.08 ms, while IR's peak is at 0.14 ms. The distributions for both methods decay to zero well below the critical value of 1 ms, shown also as a bold vertical red line at the rightmost side of Fig. 3.

4.3 Experiment 2—Force Rendering. Evaluation of force rendering is essential to determine the quality of force feedback provided by a haptic algorithm. Force discontinuities and fall-through, where a collision is not properly detected, are among the major causes of improper force feedback perceived by the user. In order to quantify the quality of haptic feedback for each algorithm, we have defined a metric called *force anomaly coefficient*.

4.3.1 Force Anomalies. Even though the volumetric dataset and its related polygonal models used in the experiments are relatively smooth, they pose a challenge to the algorithms under study. Some of the anomalies observed in the experiments include fall-through, force irregularities, and discontinuities, as well as the haptic device getting stuck into the surface.

We are interested in quantifying the smoothness of forces rendered, and therefore, the derivative of the recorded forces is used for this purpose. In our method, the derivative of the force magnitude is computed for each case using the central differences method. Once a vector containing force derivatives is calculated, the average force magnitude and its standard deviation are computed. A force anomaly is detected for each element of the force derivative vector whose value is outside the interval $[\mu - 10 * \sigma, \mu + 10 * \sigma]$, where μ is the average of the force derivative vector and σ its standard deviation. The number of values lying outside the interval divided by the total number of force measurements for each case is what we call force anomaly coefficient.

It is important to mention that we have designed this metric specifically for smooth surfaces, such as the test case described in Appendix and shown in Fig. 1. For other less frequent cases in surgical simulation where nonsmooth surfaces are required, this metric would not be entirely appropriate as in those cases large derivatives in the force magnitude are expected. For the most frequent cases, though, we have found this metric adequately captures force discontinuities and fall-through occurrences.

Table 2 summarizes the process of obtaining the force anomaly coefficient.

Figures 4–7 exemplify force anomalies detected in different experiments. Z components of the recorded trajectory are shown in blue, whereas the detected anomalies are shown in red stars. The example shown in Fig. 4 corresponds to CHAI3D, where the haptic stylus got stuck on the surface near the end of the trajectory (right of the figure). Although there is a deviation from the prescribed trajectory, there is no actual fall-through. The irregularities in the trajectory correspond to the operator pulling the haptic stylus trying to release it from the stuck position. The system reacts with high-magnitude forces as the operator tries to pull out the stylus and return to the predefined path. The peaks in red show

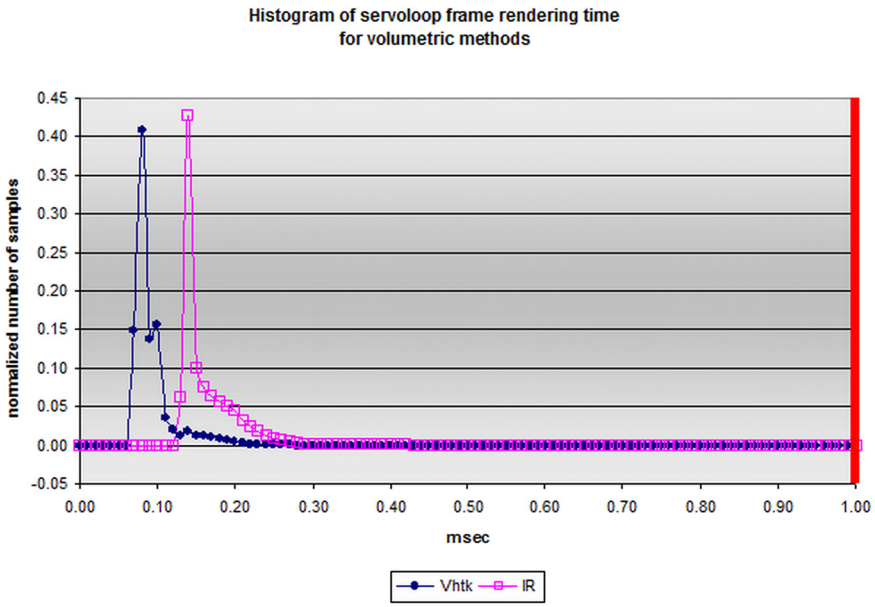

Fig. 3 Servoloop frame rendering time for volumetric methods

Table 2 Steps required to compute the force anomaly coefficient

(1) Given a vector F containing recorded forces compute |F| and calculate its derivative using the central differences method. Store the result in vector |F|′.
(2) Compute the mean and standard deviation of vector |F|′
(3) Find the values of |F|′ outside the interval $[\mu - 10 * \sigma, \mu + 10 * \sigma]$. That is the number of force anomalies
(4) Normalize the number of force anomalies by the total number of samples recorded

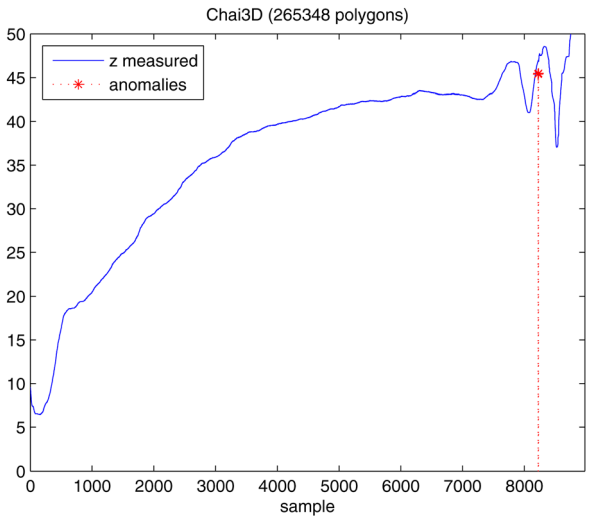

Fig. 4 Force anomalies in CHAI3D for 265K visualized polygons

anomalies detected in that particular region, where a strong force discontinuity was sensed.

On the other hand, Fig. 5 shows an actual occurrence of fall-through using the GodObject renderer in H3D, for a model consisting of 159K polygons. Force discontinuities are detected in the zone where fall-through occurs, as well as in another two regions where minor deviations from the ideal trajectory are found. Both previously discussed examples show that our evaluation method is able to identify deviations from the ideal path.

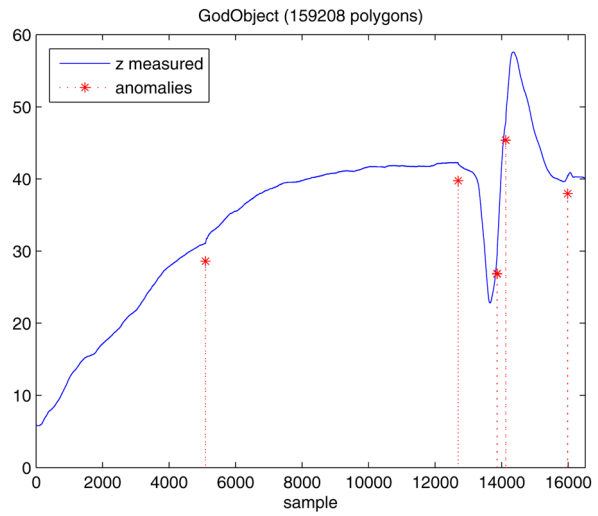

Fig. 5 Force anomalies in GodObject for 159K visualized polygons

The method also detects irregularities of rendered forces. Figure 6 presents data for the OpenHaptics renderer in H3D using Feedback Buffer, for a model consisting of 53K polygons. The operator felt certain spots along the trajectory with sudden high-level friction which are visible as small deviations from the trajectory and detected as force anomalies.

In a similar way, an experiment using VHTK is shown in Fig. 7, combined with surface graphics rendering of a model consisting of 238K polygons. Even though the friction transfer function was set to zero, the operator felt a rough surface, which is shown by small peaks in the trajectory and detected by large force variations.

From the examples shown, it is clear that the force anomaly coefficient not only detects anomalies in force rendering but also it is able to detect deviations from the trajectory through their associated force discontinuities. Thus, the method provides a good metric for assessing haptics quality.

4.3.2 Force Anomaly Coefficient for Experimental Data. In the previous subsection, some specific cases were shown. Here, we present results obtained from the whole data collected in the

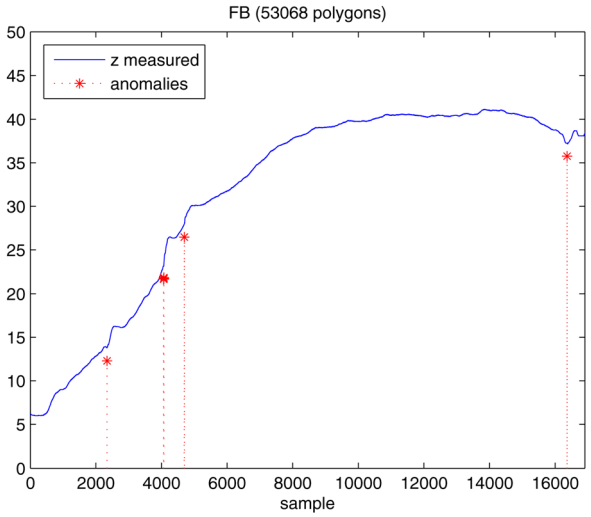

Fig. 6 Force anomalies in Feedback Buffer for 53K visualized polygons

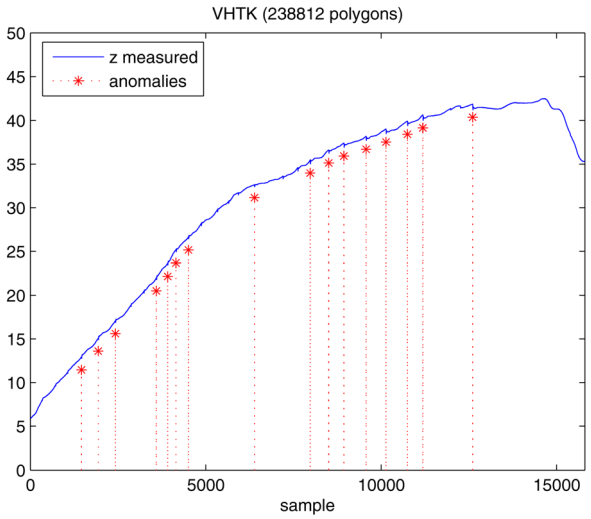

Fig. 7 Force anomalies in VHTK for 238K visualized polygons

experiment. Figure 8 shows the average force anomaly coefficient for each algorithm analyzed, as well as their maximum and minimum values. It can be seen that Depth Buffer does not show any force discontinuity or fall-through, in agreement with what the operator experienced. IR, the volume haptics algorithm implemented in the ImmersiveTouch API, also produced an absolutely smooth force feedback in all cases. All other haptics rendering algorithms presented anomalies at least once during the experiment, shown by nonzero values of their force anomaly coefficient.

4.4 Experiment 3—Client Thread Running Time. Measuring the running time of the haptics rendering code in the client thread gives us additional insight for evaluating the algorithms from a practical point of view. As shown in Sec. 4.2, almost all algorithms are able to maintain the required 1 kHz frame rate in the servo thread. However, there are substantial differences when measuring the running time in the client thread, which is executed at graphics rendering rates (30–60 Hz). This metric directly determines the usability of each algorithm, since it affects the effective interaction frame rate of the final application as perceived by the user.

Figure 9 shows the average running time of each algorithm as a function of the number of polygons in the models. It is important to remark that the number of polygons used as independent variable is common to all algorithms (even voxel-based haptic methods), as it refers to the number of polygons used for graphics visualization.

For polygonal mesh methods, the independent variable is also the number of polygons used for haptics, whereas for voxel-based haptics the model remains invariable (only the associated polygonal graphics models change). Since we are comparing the combined graphics and haptics execution times, it is still fair to compare polygon-based and volume-based haptic algorithms in a single experiment and show the results as a function of the number of polygons visualized. If a rigorous comparison among all methods is desired, one should only account for the last set of measurements in Fig. 9 (rightmost side). In that case, the polygonal mesh representing the model is directly comparable to their voxel-based counterpart, as the polygonal mesh has been obtained by the Marching Cubes from voxels and no decimation has been applied (see Appendix). All other points in Fig. 9 are presented to show dependence on polygon count for polygonal mesh-based methods.

As expected, haptics algorithms based on polygons demand more time to process the geometry as the number of polygons increase. It is also important to highlight that both voxel-based approaches (VHTK and IR) are insensitive to the number of

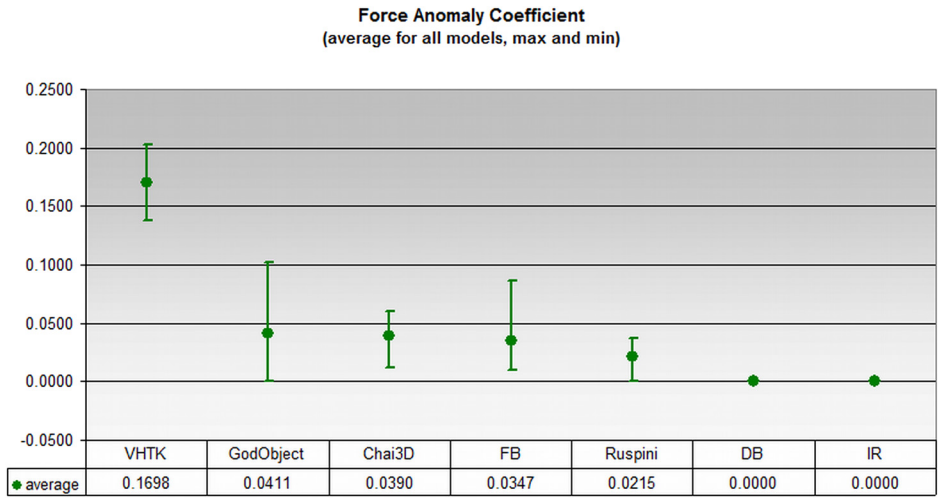

Fig. 8 Average force anomaly coefficient for all cases and its range of variation

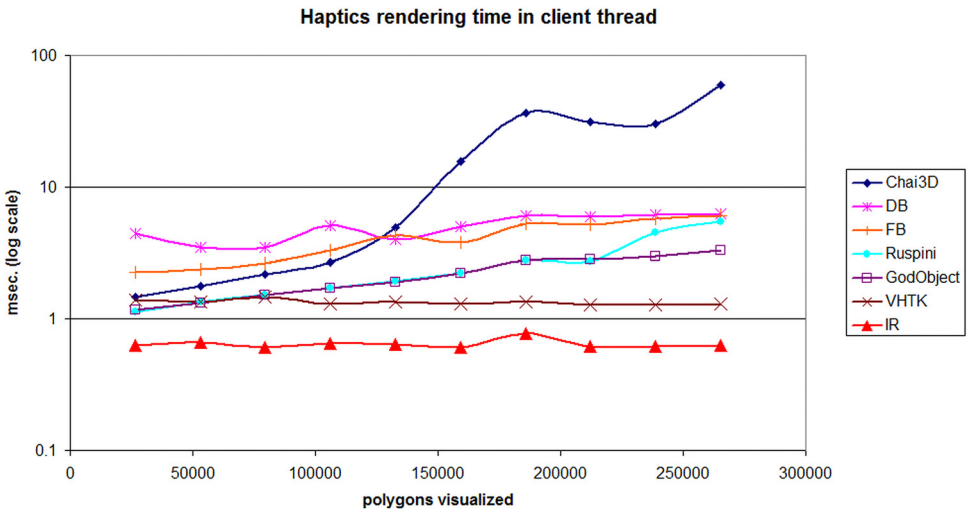

Fig. 9 Haptics rendering time in client thread

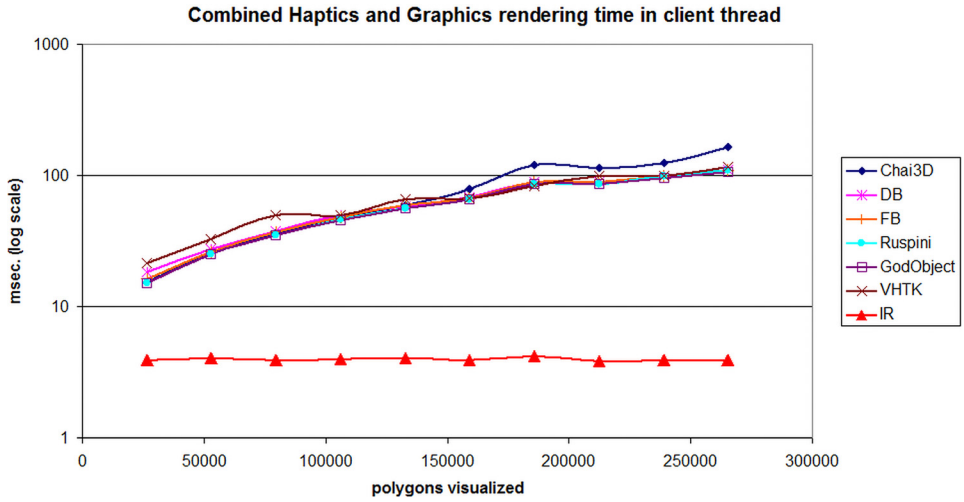

Fig. 10 Combined haptics and graphics rendering time in client thread

polygons visualized, since once the volume model is loaded in memory it is not necessary to regularly update model geometry in the client thread. Therefore, their client thread execution time for haptics updates remains essentially invariant.

Since graphics and haptics rendering in the main application thread are intimately related, we also measured the graphics rendering time for each case. As expected, the overall application frame rate is directly dependent on the combined haptics and graphics rendering time. Figure 10 shows the average combined haptics and graphics running time.

Note that all algorithms implemented in H3D have the same graphics renderer, which has a significant impact on the overall performance. In the ImmersiveTouch API, graphics rendering took an average of 3.5 ms independent of the number of polygons, the reason being a number of optimizations (Vertex Buffer Objects) present in COIN3D. In contrast, the graphics rendering time in the H3D API increases significantly with the number of polygons.

As a result, it can be seen that all combinations of haptics and graphics increase their combined execution time with the number of polygons, except for our algorithm (IR) combined with COIN3D, which maintains an almost constant execution time with 3D models of up to 270K polygons.

5 Conclusions

As previously mentioned, surgical simulation requires simultaneous graphics and haptics visualization of 3D models at interactive frame rates. It was also noted that 3D models of patient anatomy for simulation are generated from volumetric data obtained with medical imaging techniques. When complexity of the models increases, performance of the simulations is seriously affected.

One of the motivations for the work presented in this paper is the identification of haptic algorithms capable of providing adequate haptic feedback when complex isosurface models are used. For that purpose, performance of multiple algorithms in terms of rendering time and quality of haptic feedback has been experimentally evaluated in Sec. 4. The results lead to important conclusions. First of all, we found that CHAI3D, as implemented in the H3D API, is not able to maintain a minimum haptics frame rate of 1 kHz for models of a large number of polygons. This is likely the cause of undesired effects observed in experiments, such as fall-through or the haptic probe getting stuck when moving along the surface. Also, all other algorithms evaluated in this experiment were able to meet the required 1 kHz rate in the servoloop. Depth Buffer and Feedback Buffer are insensitive to the number of polygons in the models.

Table 3 Observed issues

CHAI3D	Unacceptable servoloop frame rendering times combined with the largest rendering time observed in client thread
Ruspini	Servoloop frame rendering time markedly dependent on the number of polygons and expected to become unacceptable for more than 400K polygons. Moderate force anomaly coefficient.
VHTK	Largest number of force anomalies
Depth/Feedback Buffer	Other than CHAI3D, highest client thread rendering time. Larger number of force anomalies in Feedback Buffer and higher client thread rendering time in Depth Buffer.
GodObject	Moderate number of force anomalies, lower client thread rendering time
IR	No issues observed

Naturally, VHTK and IR are also insensitive to the number of polygons for being nonpolygonal mesh-based methods. Unlike the previous algorithms, the Ruspini method shows an important dependence on the number of polygons. Extrapolating the data, we could expect that it would not maintain a 1 kHz servoloop rate for a model consisting of more than 400K polygons.

We have proposed a metric called force anomaly coefficient, which nicely captures fall-through and irregularities of force rendering. All algorithms presented some degree of irregularities during our experiments, with the exception of Depth Buffer and our volume haptics algorithm (IR) implemented in the ImmersiveTouch API.

As a result of evaluating the haptics rendering performance in the client thread, we found that the voxel-based algorithms are the fastest, the reason being that it is not necessary to regularly transmit geometry updates to the servoloop. Furthermore, when also considering graphics updates, the only combination that maintains its performance invariable is our voxel-based algorithm with COIN3D polygonal mesh graphics rendering. Based on experimental evidence, we claim that, except in our approach, the main performance bottleneck in OpenHaptics applications is to be found in the client thread. This is a point we would like to highlight and propose for further investigation.

Based on the results from Secs. 4.2, 4.3, and 4.4, we have created Table 3 where the problems observed for each algorithm are presented. We have arranged the algorithms from top to bottom according to their perceived usability (increasing downwards).

In conclusion, the combination of intermediate representation haptics in Ref. [9] with polygonal mesh visualization appears to be the most efficient method for medical simulation of highly complex models. Among all the evaluated methods, it provides the lowest total rendering time, it is insensitive to model complexity, and it correctly generates haptic feedback in all cases.

Acknowledgment

This work was supported in part by a subcontract from NIST ATP Cooperative Agreement 70NANB1H3014 and by NIH NIBIB Grant No. 1R21EB007650-01A1.

Appendix: Experimental Setup

Our experimental setup consists of a Dell Precision 690 workstation, Quad Intel Xeon @ 3.20 GHz processors, 4 GB RAM, nVidia Quadro FX 4600 graphics card, and MS Windows 7 Professional 32 bit. The haptic device is a SensAble Phantom Omni, driver version 4.2.122. Our haptics library is OpenHaptics 3.0 Academic Edition. Applications are developed using COIN3D version 3, SystemsInMotion's implementation of OpenInventor. We have also used the VHTK from SenseGraphics. In our experiments, VHTK version 1.9.0 was built along with the latest snapshot of version 2.1 of the H3D API. Version 2.0.0 of the CHAI3D library was also built and linked against the H3D API.

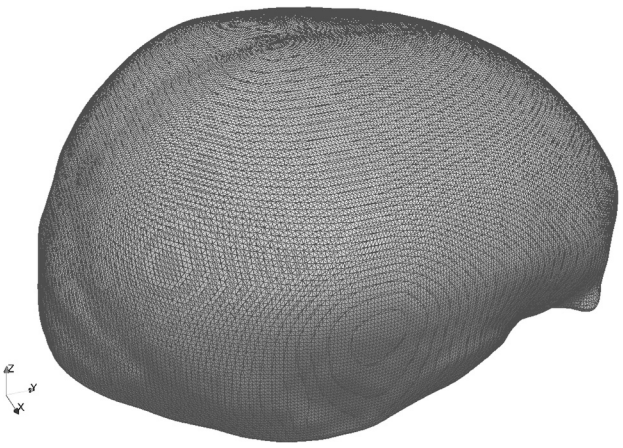

Fig. 11 Original mesh—265,348 polygons

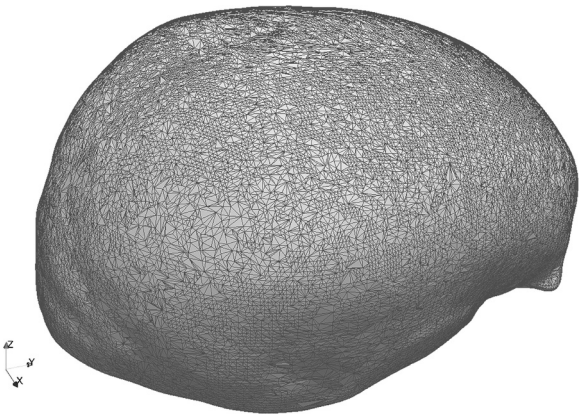

Fig. 12 50% decimated mesh—132,674 polygons

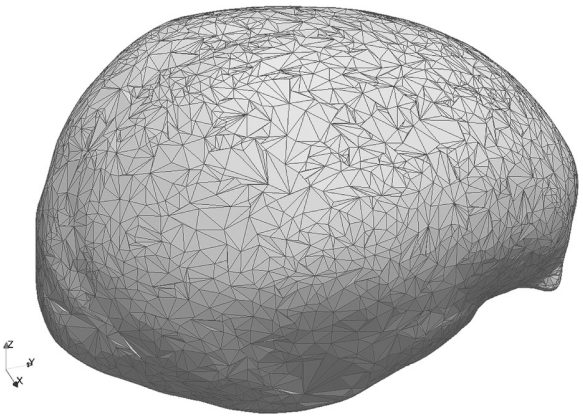

Fig. 13 90% decimated mesh—26,534 polygons

The software runs on the ImmersiveTouch platform [35,36], which is used for collocated graphics and haptics, providing also head tracking for displaying viewer-centered perspective. The display resolution is set as 1600×1200 pixels at a vertical refresh rate of 100 Hz.

The experimental dataset is a subset of a head CT scan consisting of $228 \times 234 \times 91$ voxels. Each voxel represents a volume of $0.94 \times 0.94 \times 1.25$ mm. Voxel values are linearly converted from their original DICOM representation to eight bit unsigned values. A

trapezoidal transfer function is used to extract bone surfaces from the dataset. A three-dimensional Gaussian smoothing kernel of size 7 is applied to the original data in order to smooth the edges.

The controlled variable in the experiments is the number of polygons for Surface Graphics and Surface Haptics. For this purpose, it is necessary to generate multiple polygonal meshes from the original set of voxels. The Marching Cubes [1] implementation in the Visualization ToolKit (VTK) [37] is used. In order to create models of decreasing quality, the original mesh delivered by Marching Cubes is decimated while preserving their topology using VTK's decimation algorithm. In this way, different meshes with increasing level of decimation (i.e., decreasing number of polygons) are generated from the original volumetric dataset. As a result, we obtained ten different meshes representing the same model. The number of polygons composing the meshes varies from approximately 26K (lowest quality) to 260K (maximum quality) as shown in Figs. 11–13.

A polygonal mesh extracted as an isosurface from a volumetric dataset (without decimation) is directly comparable to the isosurfaces computed on-the-fly by the voxel-based methods evaluated in this paper. Decimated polygonal meshes, having a lower polygon count than the original mesh, are expected to be less demanding in terms of computational resources.

Measurements. There are two different places where our measurements are collected in the H3D and ImmersiveTouch APIs. One is the client-application thread (also known as main-loop thread) running usually at a frequency of 30–60 Hz (in stereoscopic and monoscopic modes, respectively). The second place is within the servoloop thread, running at 1 kHz.

Measurements in the Main Application (Client) Thread. The client-application thread is managed by a GUI library generally using OpenGL Utility Toolkit (GLUT) [38] or other implementations of GLUT (e.g., FLTK). The code for haptics and graphics update is usually located in either `glutDisplayFunc()` or `glutIdleFunc()`. The callback function defined by `glutDisplayFunc()` is used to update the graphics primitives representing the geometry to be displayed. The alternative is to use the callback function defined by `glutIdleFunc()`, which is called when the computer is idle.

The evaluated APIs follow both of the above approaches. H3D updates graphics and haptics in its `Scene::idle()` method, whereas the ImmersiveTouch API uses `SoViewer::display()` for graphics updates and `SoViewer::idle()` for haptics updates. The advantage of the latter is that when graphics updates are fast, the calls to `SoViewer::idle()` are more frequent, allowing for a more frequent update of the geometry to the haptics library. Logically, we have added the timing routines for the client thread in `Scene::idle()` for H3D and `SoViewer::display()` and `SoViewer::idle()` for the ImmersiveTouch.

Measurements in the Servoloop Thread. Since forces are computed and rendered on each servoloop cycle, they must be measured in the servoloop thread. In addition, the current position of the haptic device is collected in each frame, along with the frame execution time.

In H3D, the measurement code has been placed as part of the `PhantomHapticsDevice::sendOutput()` method in `PhantomHapticsDevice.cpp`, immediately before the updated force is sent to the haptic device. In the ImmersiveTouch API, the code is part of the `computeForceCB()` callback function provided by OpenHaptics, which is also executed by the servoloop in each frame.

Measurement Techniques. In the H3D API, the current force is measured using `hdGetDoublev(HD_CURRENT_FORCE, force)`, where force is the destination variable. This function is part of the OpenHaptics HD API and it adds only a negligible overhead to the servoloop. In a similar way, the current position is measured using `hdGetDoublev(HD_CURRENT_POSITION, pos)`, where pos is the destination variable. To minimize servoloop overhead, the values collected are stored in a memory buffer, requiring only fast memory-to-memory copy operations. The buffer is dumped to a file after the experiment has concluded.

In the ImmersiveTouch API, the current force is received as a parameter in the `computeForceCB()` callback function. As in H3D, the current position of the haptics device is obtained through `hdGetDoublev(HD_CURRENT_POSITION, pos)`. These values are also stored in a memory buffer and dumped to a file at the end of the experiment.

For timing purposes in the main application thread, two functions from the MS Windows API are used, namely, `QueryPerformanceCounter()` and `QueryPerformanceFrequency()` [39]. These functions use a high-resolution performance counter with submillisecond resolution. In our system, `QueryPerformanceFrequency()` returned a value of 3,117,119, which means the system increases its counter more than three million times per second. This value represents a timer granularity of less than 1 μs, which is sufficient for our goals.

Two timestamps are taken at the beginning and the end of the haptics and graphics update routines, respectively. Two timestamps per pass allow us to calculate running time in seconds as the difference of timestamps divided by the value returned by `QueryPerformanceFrequency()`.

In contrast to the client thread, a function provided by OpenHaptics to measure frame rendering time is used in each servoloop frame. This function is `hdGetSchedulerTimeStamp()`, which returns the time elapsed since the start of the current servoloop frame [13].

References

[1] Lorensen, W., and Cline, H., 1987, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," SIGGRAPH '87 Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 163–169.

[2] Rizzi, S., Banerjee, P., and Luciano, C., 2007, "Automating the Extraction of 3D Models From Medical Images for Virtual Reality and Haptic Simulations," *IEEE International Conference on Automation Science and Engineering. CASE 2007*, pp. 152–157

[3] Systems in Motion Coin3D, http://www.coin3d.org/

[4] H3D.org, http://www.h3dapi.org/

[5] SensAble Technologies OpenHaptics, http://www.sensable.com/products-openhaptics-toolkit.htm

[6] Luciano, C., Banerjee, P., Lemole, G. M., and Charbel, F., 2006, "Second Generation Haptic Ventriculostomy Simulator Using the ImmersiveTouch™ System," *Proceedings of 14th Medicine Meets Virtual Reality*, J. D. Westwood, R. S. Haluck, H. M. Hoffman, G. T. Mogel, R. Phillips, R. A. Robb, K. G. Vosburgh, eds., IOSPress, pp. 343–348.

[7] Banerjee, P., and Charbel, F., 2006, "On-Demand High Fidelity Neurosurgical Procedure Simulator Prototype at University of Illinois Using Virtual Reality and Haptics," *Accreditation Council for Graduate Medical Education (ACGME) Bulletin*, pp. 20–21

[8] Lundin, K., 2007, "Fast and High Precision Volume Haptics," *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007*, pp. 501–506

[9] Rizzi, S., Luciano, C., and Banerjee, P., 2010, "Haptic Interaction With Volumetric Datasets Using Surface-Based Haptic Libraries," IEEE Haptics Symposium, pp. 243–250.

[10] Zilles, C. B., and Salisbury, J. K., 1995, "A Constraint-Based God-Object Method for Haptic Display," *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots,'* Vol. 3, pp. 146–151.

[11] Ruspini, D. C., Kolarov, K., and Khatib, O., 1997, "The Haptic Display of Complex Graphical Environments," *Proceedings of the 24th Annual Conference on Computer Graphics and interactive Techniques International Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, pp. 345–352.

[12] Itkowitz, B., Handley, J., and Zhu, W., 2005, "The OpenHaptics Toolkit: A Library for Adding 3D Touch Navigation and Haptics to Graphics Applications," *Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pp. 590–591

[13] SensAble OpenHaptics Toolkit Version 3.0 Programmer's Guide, Section 7.

[14] Basdogan, C., Ho, C., and Srinivasan, M. A., 1997, "A Ray-Based Haptic Rendering Technique for Displaying Shape and Texture of 3D Objects in Virtual Environments," *The Winter Annual Meeting of ASME'97*, DSC-Vol. 61, pp. 77–84.
[15] Basdogan, C., Laycock, S. D., Day, A. M., Patoglu, V., and Gillespie, R. B., 2008, "Three-Degree-of-Freedom Rendering," Haptic Rendering - Foundations, Algorithms, and Applications, Ming C. Lin and Miguel A. Otaduy, eds., A K Peters/CRC Press, Boca Raton, Fl, pp. 311–332.
[16] Lundin, K., 2007, "Direct Volume Haptics for Visualization," Ph.D. thesis, Linköping University, Linköping, Sweden.
[17] Iwata, H., and Noma, H., 1993, "Volume Haptization," *Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality*, pp. 16–23
[18] Avila, R. S., and Sobierajski, L. M., 1996, "A Haptic Interaction Method for Volume Visualization," *IEEE Visualization, Proceedings of the 7th Conference on Visualization '96*, San Francisco, CA, 197-ff.
[19] Gibson, S. F., 1995, "Beyond Volume Rendering: Visualization, Haptic Exploration, and Physical Modeling of Voxel-Based Objects," Mitsubishi Electric Research Laboratories, Technical Report No. 95-04.
[20] Petersik, A., Pflesser, B., Tiede, U., Hohn, K. H., and Leuwer, R., 2002, "Haptic Volume Interaction With Anatomic Models at Sub-Voxel Resolution," *Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems - HAPTICS* 2002, pp. 66–72.
[21] Morris, D., Sewell, C., Barbagli, F., Salisbury, K., Blevins, N. H., and Girod, S., 2006, "Visuohaptic Simulation of Bone Surgery for Training and Evaluation," IEEE Comput. Graphics Appl., **26**(6), pp. 48–57.
[22] Lundin, K., Ynnerman, A., and Gudmundsson, B., 2002, "Proxy-Based Haptic Feedback From Volumetric Density Data," *Proceedings of the Eurohaptic Conference*, University of Edinburgh, United Kingdom, pp. 104–109.
[23] Lundin, K., Gudmundsson, B., and Ynnerman, A., 2005, "General Proxy-Based Haptics for Volume Visualization," *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pp. 557–560
[24] Lundin, K., Cooper, M., and Ynnerman, A., 2005, "The Orthogonal Constraints Problem With the Constraint Approach to Proxy-Based Volume Haptics and a Solution," *Proceedings of SIGRAD Conference*, Lund, Sweden, pp. 45–49.
[25] Lundin, K., Cooper, M., Persson, A., Evestedt, D., and Ynnerman, A., 2006, "Enabling Design and Interactive Selection of Haptic Modes," Virtual Reality, **11**(1), pp. 1–13.
[26] Lundin, K., Cooper, M., and Ynnerman, A., 2008, "Haptic Rendering of Dynamic Volumetric Data," IEEE Trans. Vis. Comput. Graph., **14**(2), pp. 263–276.
[27] Adachi, Y., Kumano, T., and Ogino, K., 1995, "Intermediate Representation for Stiff Virtual Objects," *Virtual Reality Annual International Symposium*, pp. 203–210
[28] Mark, W. R., Randolph, S. C., Finch, M., Van Verth, J. M., and Taylor, R. M., 1996, "Adding Force Feedback to Graphics Systems: Issues and Solutions," *Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96*, ACM, New York, pp. 447–452.
[29] Chen, K., Heng, P., and Sun, H., 2000, "Direct Haptic Rendering of Isosurface by Intermediate Representation," *Proceedings of the ACM Symposium on Virtual Reality Software and Technology. VRST '00*, Seoul, Korea, Oct. 22-25, ACM, New York, pp. 188–194.
[30] Körner, O., Schill, M., Wagner, C., Bender, H. J., and Männer, R., 1999, "Haptic Volume Rendering With an Intermediate Local Representation," *Proceedings of the 1st International Workshop on Haptic Devices in Medical Applications*, pp. 79–84
[31] Ruffaldi, E., Morris, D., Edmunds, T., Barbagli, F., and Pai, D., 2006, "Standardized Evaluation of Haptic Rendering Systems," *14th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pp. 225–232
[32] Srimathveeravalli, G., Gourishankar, V., Kumar, A., and Kesavadas, T., 2009, "Experimental Evaluation of Shared Control for Rehabilitation of Fine Motor Skills," ASME J. Comput. Inf. Sci. Eng., **9**(1), pp. 014503-1–8.
[33] SensAble GHOST SDK Programmer's Guide, SensAble Technologies, 2002.
[34] CHAI3D.org, http://www.chai3d.org
[35] Luciano, C., Banerjee, P., Florea, L., and Dawe, G., 2005, "Design of the ImmersiveTouch™: A High-Performance Haptic Augmented VR System," Proceedings of Human-Computer Interaction (HCI) International Conference, Las Vegas.
[36] Banerjee, P., Luciano, C., Florea, L., Dawe, G., Florea, L., Steinberg, A., Drummond, J., and Zefran, M., 2010, "Compact Haptic and Augmented Virtual Reality Device," U.S. Patent No. 7,812,815.
[37] Schroeder, W., Martin, K., and Lorensen, B., 2006, *The Visualization ToolKit—An Object-Oriented Approach to 3D Graphics*, 4th ed., Kitware Inc., Clifton Park, NY.
[38] GLUT, The OpenGL Utility Toolkit, http://www.opengl.org/resources/libraries/glut/
[39] QueryPerformanceCounter Function, Microsoft MSDN library, http://msdn.microsoft.com/en-us/library/ms644904(VS.85).aspx