Haptic Interaction with Volumetric Datasets Using Surface-based Haptic Libraries

Silvio H. Rizzi

Department of Mechanical and Industrial Engineering, University of Illinois-Chicago

Cristian J. Luciano

Department of Mechanical and Industrial Engineering, University of Illinois-Chicago

P. Pat Banerjee

Departments of Mechanical and Industrial Engineering, Computer Science, Bioengineering, University of Illinois-Chicago

ABSTRACT

An algorithm to provide haptic feedback directly from volumetric datasets is introduced in this paper. The algorithm is able to reproduce and extend the functionality of surface-based haptics rendering methods. It can be easily implemented on top of existing haptic libraries, such as Sensable's OpenHaptics. In addition, our approach overcomes poor performance in OpenHaptics for models consisting of a large number of polygons (1Million+). The algorithm allows detecting collisions with multiple 3-dimensional shapes. After comparing alternative approaches, such as Depth Buffer and Feedback Buffer, as well as SenseGraphics' Volume Haptics ToolKit (ScalarSurfaceFriction mode), our algorithm delivers the highest performance in terms of rendering time.

1 INTRODUCTION

Lately, volumetric data sets have acquired extraordinary significance in medical simulation. Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) are examples of ubiquitous technologies from which 3-dimensional (3D) data sets are obtained. 3D computer models are commonly generated for Virtual Reality and Haptics simulation of medical and surgical procedures [1], [2].

Traditionally, a combination of scene graph managers [3], [4] and haptic libraries [5] is used in simulations for simultaneous graphics and haptics rendering of 3D models. Those libraries commonly require 3D objects to be represented as polygonal meshes, i.e. surfaces in 3D space consisting of multiple triangles. These polygonal meshes are usually generated using an isosurface extraction algorithm such as Marching Cubes [6]. Further processing may be required in order to reduce the number of triangles in each model (decimation) and to obtain smooth surfaces. All these processing stages often demand several hours - or even days- to complete, requiring the use of additional software tools and a considerable amount of human intervention to generate high quality 3D models. Although methods to accelerate and improve the degree of automation of the segmentation process have been discussed in the literature [7], alternative approaches must be explored in order to improve the simulations.

OpenHaptics [5] is one of the most popular commercial haptic libraries supporting SensAble haptic devices [8], [9]. It is extensively used in a number of systems for Haptics and Virtual and Augmented Reality applications, including H3D [4] and OpenHaptics-enabled versions of the software described in [10],[11]. It has, however, serious limitations when it is required to haptically render highly complex shapes. Its two modes (Feedback Buffer and Depth Buffer) impose their own constraints on the model to be rendered. On one hand, Feedback Buffer delivers high quality haptic rendering, however its performance is dependent on the number of polygons in the model. On the other hand, Depth Buffer is insensitive to the number of polygons, but there are some cases where it exhibits "noticeable discontinuities when feeling shapes with deep, narrow grooves or tunnels" [12].

To overcome the limitations discussed above, a more natural and straightforward approach would be to implement a direct volume haptics algorithm. Volumetric data could then be used directly as delivered by imaging systems, reducing or even eliminating the need of preprocessing stages to build models as polygonal meshes. Furthermore, a robust direct volume haptics could be a viable alternative to address those problems where OpenHaptics fails.

In this work an algorithm for volume haptics based on proxy methods for point-based haptic interaction is presented. This algorithm allows the use of 3D datasets without computationally expensive pre-processing stages. In addition, the algorithm is able to detect collisions based on the proxy position. As in surface-based haptics, the touching face is also determined, detecting whether the probe is at either side (back or front) of the colliding shape. The performance of the algorithm is evaluated in terms of rendering time using representative datasets, and is compared to other existing haptics approaches.

2 LITERATURE REVIEW AND RELATED WORK

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, the god-object algorithm, used for single-point contacts, was proposed in [13]. This method uses a "god-object" to constrain the haptic interface point to the mesh surface, avoiding penetration. The tip of the haptic device is coupled to the proxy through a spring model. In each haptic frame the force rendered is 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 [14]. The same paper also proposes 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], [15].

2.2 Volume Haptics Rendering

In volume haptics rendering, force feedback is generated from volumetric datasets. Iwata and Noma present in [16] an approach called Volume Haptization. For scalar data, they proposed mapping voxel values to torque vectors or mapping the gradient of

Email: {srizzi2,clucia1,banerjee}@uic.edu

voxel values to force vectors. Avila and Sobierajski [17] describe 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.

Volume haptics has been systematically studied in a series of publications [18], [19], [20], [21]. In [18], a method to generate surface and viscosity haptic feedback from volumes, along with simulation of material properties is presented. The method evolved in [19], where haptic primitives, such as directed force, point, line, and plane were presented as building blocks of 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 is refined in [20], which describes a numeric solver to compute the final forces. In [21], the Volume Haptics ToolKit (VHTK) is presented and implemented as an extension to SenseGraphics H3D [4] application programming interface (API).

2.3 Intermediate Representation methods

Intermediate representation methods were first proposed in [22]. The idea consists of representing touchable surfaces at a given point by a virtual plane tangent to the surface at that point. The collision detection loop runs independent of the servoloop and is updated at a lower rate, whereas the servoloop is updated at a higher rate required to render stiff objects. Combining intermediate representations and lower update rates simplifies the collision detection problem and helps to quickly detect collisions between the tip of the haptic device and the virtual plane. The method, however, has a fundamental limitation. If the update rate for the virtual plane is too low, the operator will perceive discontinuities as the proxy "jumps" from one plane to another. This problem was addressed in [23], where the recovery time method was presented. The method reduces the magnitude of the force immediately after a new virtual plane is 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 [24]. The algorithm extracts virtual planes from the volumetric data without the need of precomputing isosurfaces. This algorithm, combined with a proxy-based method, allows to generate haptic feedback directly from the volumetric data. However, when the proxy position lies on one side of a thin object and the haptic device is moved to a point on the other side of the thin object, then collision with the object is ignored. If this happens, the virtual plane is not computed, no force is generated, and the proxy does not stop on the surface of the haptic object. This issue is exacerbated when the haptic device is moved at relatively higher speeds. In [25], an intermediate local representation which uses Marching Cubes to generate isosurfaces from voxel data adjacent to the haptic stylus position is proposed. Local isosurfaces from a 7x7x7 cube are passed to the haptics loop in GHOST as an intermediate representation of the local volume data.

3 METHODOLOGY

3.1 Overview

Our algorithm is based on proxy methods and is inspired by the intermediate representation method described in [24]. It essentially consists of detecting collisions between the proxy point and one or more 3D shapes representing objects of interest. Shapes are defined from a set of voxels using transfer functions without the need to generate polygonal meshes.

The algorithm receives two points as parameters (`Start` and `End`) for each haptic frame rendered by the servoloop at 1 KHz. The `Start` point is the proxy position calculated in the previous haptic frame whereas the `End` point is the current position of the haptic device.

The collision detection routine detects the intersection between a line segment (determined by `Start` and `End`) and a shape surface (Figure 1). Shape surfaces are defined in terms of voxel intensities, similar to isosurfaces. The algorithm returns the 3D coordinates of the intersection point P, the normal vector N of the surface at the intersection point, and the touched side (front or back) of the shape surface. It also returns TRUE if there is a collision or FALSE otherwise. With this information, the underlying haptic library computes the forces as in the case of polygonal mesh haptics, and positions the proxy at the point P when a collision with the shape is detected.

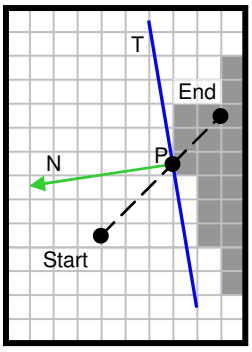

Figure 1. Problem geometry

3.2 Algorithm Details

Figure 2 shows a flow diagram of the algorithm. If the haptic stylus has not moved in two successive haptic frames and there was no collision in the previous frame, then the Start and End points are exactly the same and therefore, the function returns FALSE. Otherwise, it continues with a rough bounding-box comparison between the line and the volume, quickly returning false if they are disjoint.

If the line is inside the volume bounding box, for each point P on the line segment from the Start to the End points, the algorithm checks the intensity to the closest voxel V by a set of window transfer functions defining the multiple shapes. If the voxel intensity is outside the windows specified through the transfer functions, then the haptic device has not yet collided with any shape and the loop continues with the next point. If none of the points on the line segment collide with any shape, the function returns false.

In case the intersected voxel V lies within any of the transfer function windows, the algorithm returns the 3D coordinates of the point P as the surface contact point. The density of the voxel V is used to determine which of the shapes has been touched by the haptic device by comparing it with the ranges defined by their transfer functions.

The normal vector N, which is perpendicular to the volumetric isosurface at the contact point P, is determined by computing the gradient of the neighbor voxels using the central differences method. The contact point P and the normal vector N define a plane T (tangential to the shape) which serves as an intermediate representation of the isosurface. This plane is useful to determine if the haptic device is touching either the front or back side of the shape. If the Start point is in front of the plane T and the End point is behind it, then the colliding face is front. Otherwise, the colliding face is back. In this case, the algorithm inverts the direction of the previously computed normal vector N.

3.3 Algorithm parameters

There are two essential design requirements for this algorithm. First, it must be efficient in the sense that it must not affect the haptics servo loop sustaining a minimum 1 KHz haptic frame rate. Second, it must be robust to avoid undetected collisions. Both requirements are directly affected by the selection of the step size

with which successive discreet points along the line segment (shown in Figure 1) are evaluated. If the step size is too big, a collision could be overlooked, especially for thin structures. On the other hand, a finely grained step size could help guarantee collisions are always detected, but it could also severely impact the haptic frame rate, since the algorithm is executed in the servo loop thread.

If we parameterize the line segment from Start to End with parameter *i*, where *i* is in the interval [0,1], then the following linear interpolation equation gives any point P in the line segment as a function of *i*:

$$\mathbf{P} = (1 - i) \cdot \mathbf{Start} + i \cdot \mathbf{End} \quad (1)$$

Our algorithm traverses the line segment by varying *i* from 0 to 1, incrementing it by a value of `delta` in each successive iteration. Computations of P are done in continuous space and further converted to discrete voxel coordinates for retrieving voxel values. No sub-voxel resolution is needed.

Users can move the haptic stylus at various speeds, which is reflected in corresponding variations of the line segment length from `Start` to `End`. Therefore, `delta` must be carefully selected each time the algorithm is executed. A naïve approach

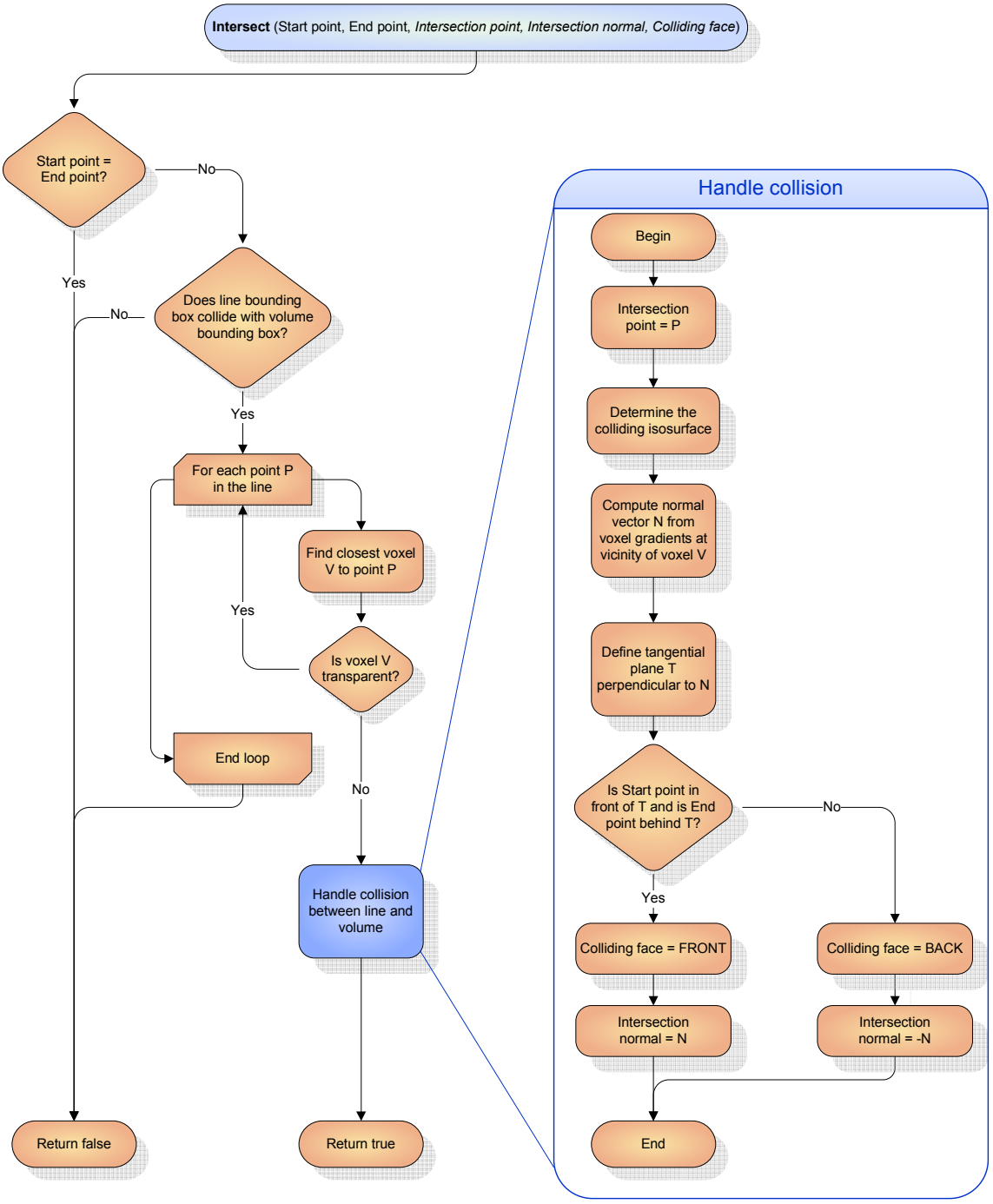

Figure 2. Flow diagram of the algorithm

would be to divide the line segment into a constant number of steps, so the number of iterations is constant for all moving speeds. However, this approach would fail to detect collisions when the haptic device is moved at high speeds, especially when structures are thin. The problem is solved using a variable step size as follows:

$$delta = \frac{k}{|\mathbf{End} - \mathbf{Start}|} \qquad (2)$$

where k is a constant that depends on the voxel size. In this way, for higher speeds, the interval [0,1] representing the line segment is divided into a higher number of steps. As shown in Figure 2, when the haptic device does not move in two successive haptic frames (`Start = End`), the algorithm returns immediately, preventing division by zero in Equation (2). Every time the algorithm is executed, the actual distance between successive points P to be evaluated in a given line segment is constant, regardless of the velocity of the haptic stylus. Thus, initializing k to be equal to or less than the minimum dimension of voxels is a necessary condition to prevent undetected collisions.

3.4 Transfer Functions

The algorithm is able to simultaneously detect multiple shapes from volumetric isosurfaces defined by their individual ranges of voxel intensities. A transfer function is defined for each haptic shape whereby a binary output value is assigned to each possible voxel intensity. In graphics volume rendering techniques, piece-wise linear transfer functions are commonly used to specify color intensities and transparency. Similarly, in our approach transfer functions are used to determine whether a voxel should be touchable or not based on its intensity.

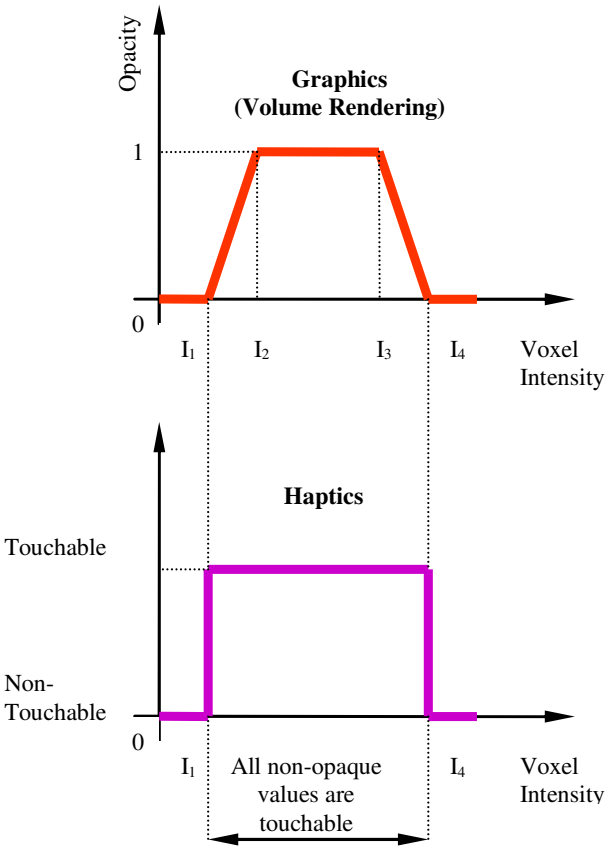

Figure 3. Comparison of Graphics and Haptics transfer functions

Figure 3 presents a comparison between graphics volume rendering and haptic transfer functions. The first transfer function exemplifies opacity as a function of voxel intensities. Gradually increasing or decreasing values of opacity, represented by ramps, are allowed and commonly used. On the other hand, in the haptics transfer functions only discrete binary outputs are permitted. In this way, voxels with intensities within the rectangular window defined by the transfer function will be regarded as belonging to the shape and will, therefore, be touchable. In other words, when the collision detection algorithm finds a voxel whose intensity is within the rectangular window, it will return TRUE, indicating a collision with the shape was detected.

There are two advantages to using haptic transfer functions. First, since they are similar to the ones commonly used in volume visualization techniques, a single transfer function may simultaneously specify graphics and haptics properties for each shape. In Figure 3 it is shown how a haptic transfer function can be obtained from its graphics counterpart. As a result, all non-opaque values will be touchable and haptic parameters such as stiffness, static friction, and dynamic friction will be assigned to the corresponding voxels. The second advantage is that pre-processing steps such as segmentation and construction of polygonal meshes for each shape are no longer needed. In essence, the haptic transfer functions implemented resemble an operation of binary thresholding, by which different subsets may be determined from the original dataset according to their voxel intensities. Therefore, the specification of transfer functions provides all the information needed to generate graphics and haptics visualization, operating only with the original (unmodified) 3D dataset.

3.5 Implementation

Our algorithm is intended to take advantage of the efficient force computation implemented in existing haptic libraries. Therefore, the algorithm performs the collision detection and passes to the haptic library all the information needed to compute the forces in the same way it does for polygonal meshes.

OpenHaptics allows users to define custom shapes by a callback function which is called in each frame of the servoloop thread, before computing the forces to be sent to the haptic device. The prototype of the intersect callback function for OpenHaptics [26] is as follows:

```
bool intersectSurface(const HLdouble startPt[3],
                      const HLdouble endPt[3],
                      HLdouble intersectionPt[3],
                      HLdouble intersectionNormal[3],
                      HLenum   *face,
                      void     *userdata  );
```

Our algorithm, implemented as a callback function responding to the `intersectSurface` prototype, returns the coordinates of the contact point P, the intersection normal vector N, and the touching face (as the third, fourth and fifth parameters, respectively). OpenHaptics computes forces based on the haptic materials associated with the haptic shape (spring, damper, static and dynamic friction), allowing the user to feel the contact and friction between the proxy and the volumetric isosurfaces. Similar to the case of polygonal meshes, by setting the haptic shape's touchable face as HL_FRONT, HL_BACK or HL_FRONT_AND_BACK, the algorithm allows the user to feel only one or both sides of the haptic isosurface. If there is no collision, OpenHaptics updates the proxy position with the current position of the haptic device. On the other hand, if there is a

collision, OpenHaptics fixes the proxy at the surface contact point P, and computes the forces to be sent to the haptic device.

To detect collisions with multiple shapes, OpenHaptics calls the `intersectSurface` callback function once for each haptic shape defined. Those multiple calls are made within each individual servo frame at 1 KHz. If the `intersectSurface` callback function returns TRUE, there is a collision with the current shape. It returns FALSE, otherwise. However, the algorithm needs to be executed only once per haptic frame since the goal is to find the first non-transparent voxel along the line segment from Start to End. The sixth argument (`*userdata`) in the `intersectSurface` prototype is used to pass the shape to be evaluated in each call. The collision detection algorithm is executed only during the call corresponding to the first shape in a servo frame. If a collision is detected, the algorithm will determine the touched shape comparing the density value of the voxel against the value ranges of the transfer function. After the first call within a servo frame, there is no need to execute the collision detection algorithm again. However, based on the `*userdata` parameter, the callback function will return TRUE when the call corresponds to a collided shape, and FALSE for all the other shapes. In this way, OpenHaptics' `intersectSurface` callback function is called multiple times (once per existing shape) but the collision detection algorithm is executed only once in each frame of the servo loop, obtaining a constant runtime independent of the number of shapes.

4 Experimental Setup And Results

4.1 Experimental Setup

Our setup consists of a Dell Precision 690 workstation, Quad Intel Xeon @ 2.66 GHz processors, 4 GB RAM, two nVidia QuadroFX 4600 graphics cards in SLI configuration, and MS Windows XP Professional Service Pack 2. The haptic device is a SensAble Phantom Desktop, driver version 4.2.49. Our haptic library is OpenHaptics 2.0 Academic Edition. The experimental applications were developed using Coin3D 2.5.0, SystemsInMotion's implementation of OpenInventor. SIMVoleon version 2.0.1, also from SystemsInMotion, is used for volume rendering. To test our implementation against other solutions, we have used the Volume Haptics ToolKit (VHTK) from SenseGraphics. In our experiments, VHTK version 1.5.1 was built on top of version 1.5 of the H3D API. VHTK includes its own implementation of graphics volume rendering.

The software runs on the ImmersiveTouch platform [10], [11], which is used for collocated graphics and haptics, providing also head tracking for viewer's centered perspective. The display resolution is set as 1600x1200 pixels with a vertical refresh rate of 100 Hz with quad buffering for stereoscopic visualization.

For the experiments a CT dataset from a patient's head is used, consisting of 512x512x192 voxels linearly converted from DICOM files to 8 bits per voxel. Voxel sizes are 0.48x0.48x1.25 millimeters. A preset value of k = 0.1 mm is used in Equation (2). Since the minimum voxel dimension is 0.48 mm, the preset value of k prevents undetected collisions. A 3D Gaussian smoothing kernel of size seven is applied to the original data in order to smooth edges.

4.2 Performance Comparisons

The objective of this experiment is to measure the real-time performance of the presented algorithm in an environment whose conditions are similar to an actual application, comparing it to OpenHaptics' Depth and Feedback Buffer methods for polygonal meshes, and also to the ScalarSurfaceFrictionMode in VHTK. For that purpose, different combinations of graphics and haptics rendering modes are tested, as shown in the following table:

[DB]	Surface Graphics using OpenInventor and Surface Haptics using OpenHaptics' Depth Buffer
[FB]	Surface Graphics using OpenInventor and Surface Haptics using OpenHaptics' Feedback Buffer
[SVHTK]	Surface Graphics using H3D and Volume Haptics using VHTK
[VVHTK]	Volume Graphics and Volume Haptics both using VHTK
[SV]	Surface Graphics using OpenInventor and our algorithm for Volume Haptics
[VV]	Volume Graphics using SimVoleon and our algorithm for Volume Haptics

For Volume Graphics, the number of slices for the volume renderer implementations varies from 200 to 2000 slices. For Surface Graphics, it is necessary to generate a polygonal mesh from the original set of voxels. For this purpose, the Marching Cubes implementation in the Visualization ToolKit (VTK) is used. In this case, the number of polygons in the polygonal mesh varies from approximately 100k to 1200k polygons. The number of polygons is modified while preserving the mesh 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 mesh.

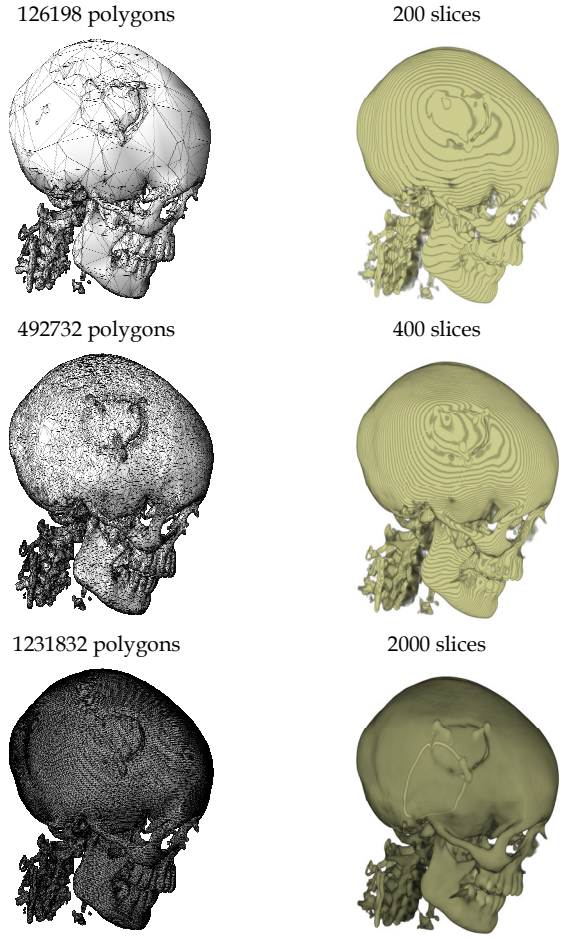

Figure 4. Polygonal mesh including wireframe (left) and volume (right) renderings of the model used for experiments.

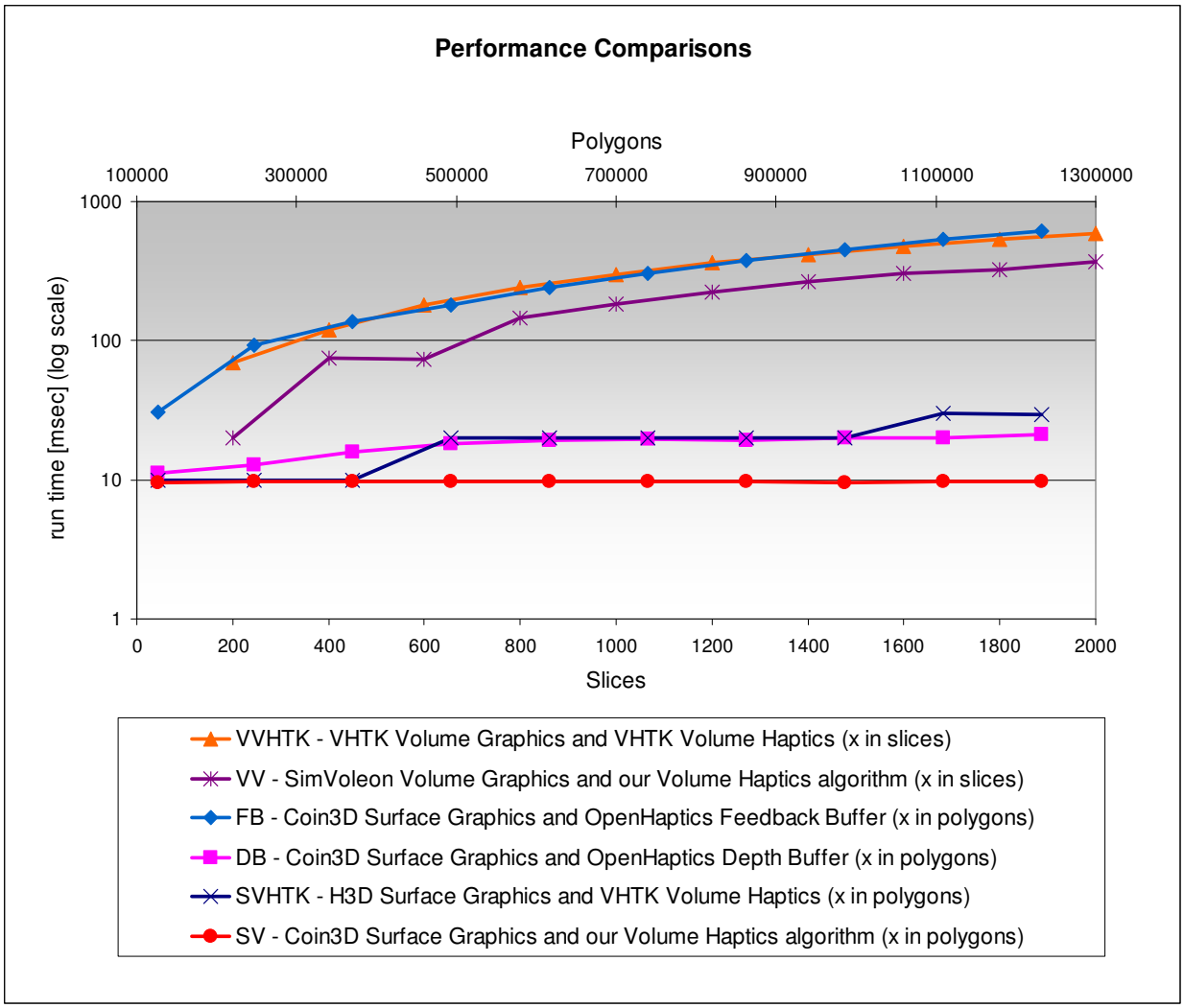

Figure 5. Results of performance comparisons.

Figure 4 presents different renderings of the model used for the experiments for both Surface Rendering and Volume Rendering. For Surface Rendering, the wireframe is also visualized to give a clear idea of its polygonal density. Note that the models vary from low to very high quality, depending on either the number of polygons or number of slices, which is intended to cover the practical range of actual applications.

The results of the experiments for all test cases are presented in Figure 5. The measurements reflect the time elapsed in each frame of the main rendering loop, including both graphics and haptics rendering. These measurements are an indication of how long it takes to fully render an entire frame in the main application thread, and they are, thus, a good measure of actual performance of an application.

By OpenHaptics' API design, it is not possible to obtain direct measurements of the execution time of each servo loop frame (in the haptics thread). It is, however, possible from the main application thread to request OpenHaptics' HD API the average and instantaneous servo loop rates. In all the experiments OpenHaptics maintained its instantaneous servo loop rate (measured using `hdGetIntegerv( HD_INSTANTANEOUS_UPDATE_RATE, &rate)` from the application's main loop) at about 1000 Hz. Given that measurement we must assume that no servo loop frames are dropped for all the experiments conducted.

The technique for measuring run time is implemented in such a way that values for all cases are comparable, even though they involve substantially different algorithms, libraries, and APIs. In cases where VHTK is used, minimum modifications were introduced in one of H3D's source files (file Scene.cpp, method Scene::idle()) in order to compute its combined rendering time.

5 Discussion

From the experiments, it can be seen that the performance of [FB], [VV], and [VVHTK] deteriorate as the number of polygons or slices is increased. This is the expected behavior for [FB], since OpenHaptics' Feedback Buffer algorithm depends on the number of polygons. Similarly, for both graphics volume rendering implementations (SimVoleon in [VV], VHTK in [VVHTK]) the performance becomes unacceptably low in terms of graphics frame rate as the number of slices for the given dataset is increased.

On the other hand, [DB] using OpenHaptic's Depth Buffer shows an initially decreasing performance which stabilizes as the number of polygons increases. Since the visualization load for [DB] is the same as for [FB], differences in performance must be attributed to the use of different haptics algorithms. Similarly,

[SVHTK] using VHTK for haptics volume rendering and H3D's capabilities for polygonal mesh rendering shows an initially high value followed by two different plateau regions with steep transitions between them. The haptics load is the same for [VVHTK] and [SVHTK], and therefore their different behaviors are due to their visualization schemes, where polygonal mesh visualization outperforms its graphics volume rendering counterpart. In addition, performance is comparable for [DB] and [SVHTK] in the middle region (approx. 500k to 1000k polygons), whereas [DB] outperforms [SVHTK] for a higher number of polygons.

Finally, [SV] combining our volume haptics algorithm and polygonal mesh visualization exhibits an almost flat performance in its entire range, outperforming all other cases. The visualization load for cases [DB], [FB], and [SV] is exactly the same. Therefore, based on the above reasoning, the difference in the performance lies in their haptics implementations.

From these results, it is clear that our volume haptics algorithm delivers the highest and most stable performance when compared, under similar conditions, to OpenHaptics' Depth Buffer and Feedback Buffer implementations. Besides the OpenHaptics library, there is no shared code between our applications ([DB], [FB], [SV], and [VV]) and VHTK ([SVHTK] and [VVHTK]) . However, the use of exactly the same dataset and measuring techniques suggests that our algorithm also outperforms VHTK's Scalar Surface Friction Mode for this specific task.

Comparing our algorithm with previous intermediate representation approaches, our major contributions are:

1) Elimination of inadequate haptic feedback: as discussed in Section 2.3, there is a limitation in [22] where a lower update rate for the intermediate representation with respect to the servoloop rate may cause irregularities in the force feedback. The problem is contemplated in the recovery time approach [23], but not eliminated. In [24], the update rate of the intermediate representation is 1/n of the force computation rate, and so the problem in [22] is also present whenever $n > 1$. In our approach, collisions are detected at exactly the same rate in which the servoloop is updated, thus each execution of the collision detection is guaranteed to precede the force computation. Therefore, our algorithm eliminates this problem inherent in intermediate representations.

2) No fall-through for thin structures: The algorithm in [24] may fail to detect collisions with thin structures, as explained in Section 2. This problem is not present in our algorithm, where the speed at which the haptic device is moved does not affect the robustness of the collision detection algorithm, as shown in Section 3.3.

3) Haptic front/back face detection: Building our algorithm on top of an existing haptic library allows detection of back/front faces and to assign different haptic properties to each one. This is not possible in [22] and [24]

4) Multiple shape detection: Our algorithm is implemented such that it is possible to efficiently detect multiple shapes and assign different haptic properties to each of them. This feature is not discussed in [22] and [24].

5) Leveraging of existing libraries: Building our algorithm as part of an existing haptic library allows one to use volumetric as well as polygonal mesh models at the same time. Moreover, there are additional advantages from using the OpenHaptics library that come for free, such as pop-through effects as well as touch/untouch callback functions.

In summary, we have introduced an algorithm to provide haptic feedback directly from volumetric datasets. It overcomes poor performance in OpenHaptics for models consisting of a large number of polygons and allows detecting collisions with multiple 3-dimensional shapes. Our algorithm delivered the highest performance in experimental comparisons with Depth Buffer, Feedback Buffer, and the ScalarSurfaceFriction mode in VHTK.

ACKNOWLEDGMENT

This work was supported in part by NIH NIBIB grant 1R21EB007650-01A1.

REFERENCES

[1] C. Luciano, P. Banerjee, G.M. Lemole, F. Charbel, "Second Generation Haptic Ventriculostomy Simulator Using the ImmersiveTouch™ System," Proceedings of 14th Medicine Meets Virtual Reality, J.D. Westwood et al. (Eds.), IOSPress, pp. 343-348, 2006.

[2] P. Banerjee, F. Charbel, "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, September 2006; p. 20-21.

[3] Systems In Motion Coin3D, available at http://www.coin3d.org/

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

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

[6] W. Lorensen, H. Cline, "Marching Cubes: A high resolution 3D surface construction algorithm," Computer Graphics, Vol. 21, No. 4, July 1987.

[7] S. Rizzi, P. Banerjee, C. Luciano, "Automating the Extraction of 3D Models from Medical Images for Virtual Reality and Haptic Simulations," Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on , vol., no., pp.152-157, 22-25 Sept. 2007

[8] T.H. Massie, J.K.Salisbury, "The PHANTOM Haptic Interface: A Device for Probing Virtual Objects," Symp. On Haptic Interfaces for Virtual Environments. Chicago, IL, Nov. 1994.

[9] SensAble Haptic Devices, available at http://www.sensable.com/products-haptic-devices.htm

[10] C. Luciano, P. Banerjee, L. Florea, G. Dawe, "Design of the ImmersiveTouch™: A High-Performance Haptic Augmented VR System," Proceedings of Human-Computer Interaction (HCI) International Conf. Las Vegas, 2005.

[11] P. Banerjee, C. Luciano, L. Florea, G. Dawe, et. al, Compact haptic and augmented virtual reality device. U.S. Provisional Patent Application no. 60/646,837, March 2005, U.S. Patent Application, January 2006.

[12] SensAble OpenHaptics Toolkit Version 2.0 Programmer's Guide, pp. 6-6

[13] C.B. Zilles, J.K. Salisbury, "A constraint-based god-object method for haptic display," Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on , vol.3, no., pp.146-151 vol.3, 5-9 Aug 1995

[14] D.C. Ruspini, K. Kolarov, O. Khatib, "The haptic display of complex graphical environments". In 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, NY, 345-352, 1997.

[15] B. Itkowitz, J. Handley, W. Zhu, "The OpenHaptics toolkit: a library for adding 3D Touch navigation and haptics to graphics applications," Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005, pp. 590-591, 18-20 March 2005.

[16] H. Iwata, H. Noma, "Volume haptization," Virtual Reality, 1993. Proceedings., IEEE 1993 Symposium on Research Frontiers in , vol., no., pp.16-23, 25-26 Oct 1993

[17] R.S. Avila, L.M. Sobierajski, "A haptic interaction method for volume visualization." In Proceedings of the 7th Conference on Visualization '96 (San Francisco, California, United States, October 28 - 29,

1996). R. Yagel and G. M. Nielson, Eds. IEEE Visualization. IEEE Computer Society Press, Los Alamitos, CA, 197-ff.

[18] K. Lundin, A. Ynnerman, B. Gudmundsson, “Proxy-based haptic feedback from volumetric density data.” In Proceedings of the Eurohaptic Conference, pp. 104-109. University of Edinburgh, United Kingdom, 2002.

[19] K. Lundin, B. Gudmundsson, A. Ynnerman, "General proxy-based haptics for volume visualization," Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005, pp. 557-560, 18-20 March 2005.

[20] K. Lundin, M. Cooper, A. Ynnerman, “The orthogonal constraints problem with the constraint approach to proxy-based volume haptics and a solution.” In Proceedings of SIGRAD Conference, pp. 45-49, Lund, Sweden, Nov. 2005.

[21] K. Lundin, M. Cooper, A. Persson, D. Evestedt, A. Ynnerman, “Enabling design and interactive selection of haptic modes.” Virtual Reality, 2006. DOI: 10.1007/s10055-006-0033-7.

[22] Y. Adachi, T. Kumano, K. Ogino, "Intermediate representation for stiff virtual objects," Virtual Reality Annual International Symposium, 1995. Proceedings, pp.203-210, 11-15 Mar 1995.

[23] W.R. Mark, S.C. Randolph, M. Finch, J.M. Van Verth, and R.M. Taylor, “Adding force feedback to graphics systems: issues and solutions”. In Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96. ACM, New York, NY, 447-452. 1996.

[24] K. Chen, P. Heng, and H. Sun, “Direct haptic rendering of isosurface by intermediate representation.” In Proceedings of the ACM Symposium on Virtual Reality Software and Technology (Seoul, Korea, October 22 - 25, 2000). VRST '00. ACM, New York, NY, 188-194.

[25] O. Körner, M. Schill, C. Wagner, H.J. Bender, R. Männer, “Haptic volume rendering with an intermediate local representation”. In: Proc of the 1st International Workshop on Haptic Devices in Medical Applications, pp. 79–84 (1999)

[26] SensAble OpenHaptics Toolkit Version 2.0 Programmer’s Guide, pp. 6-32