Haptic Rendering of Mixed Haptic Effects

Lei Wei and Alexei Sourin
School of Computer Engineering
Nanyang Technological University
Singapore

e-mail: WEIL0004@e.ntu.edu.sg, assourin@ntu.edu.sg

Abstract—Commonly, surface and solid haptic effects are separated for haptic rendering. We propose a method for defining surface and solid haptic effects as well as various force fields in 3D cyberworlds containing mixed geometric models, including polygon meshes, point clouds, image-based billboards and layered textures, voxel models and functions-based models of surfaces and solids. We also propose a way how to identify location of the haptic tool in such haptic scenes as well as consistently and seamlessly determine haptic effects when the haptic tool moves in the scenes with objects having different sizes, locations, and mutual penetrations.

Keywords- haptic interaction, collision detection, function-based modeling

I. INTRODUCTION

Haptics is a term which refers to the incorporation of touch sense into computer programs by providing force feedback which is delivered by haptic devices. The whole procedure from user manipulation to the final device feedback is called haptic interaction.

There are various kinds of haptic devices available for many different research areas. The devices can be classified by the number of Contact Points (CP) they have. Some haptic devices, which are mostly used in research works, provide only one contact point. They are designed as a stylus attached to a robotic arm allowing for navigating a virtual instrument in a 3D space with 3 or 6 degrees of freedom (DOF) and a variable force-feedback. Depending on the output DOF, these single contact-point haptic devices can be further classified as 3DOF force devices and 6DOF force devices, which render only one force vector or one force vector plus one torque vector as the force feedback, respectively. Examples of such devices are those produced by Sensable Technology (http://www.sensible.com), Immersion (http://www.immersion.com), Force Dimension (http://www.forcedimension.com), Quanser (http://www.quanser.com), MPB Technologies (http://www.mpb-technologies.ca), ERGOS Technologies (http://acroe.imag.fr/ergos-technologies), Haption (http://www.haption.com), and Novint (http://home.novint.com). Other haptic devices such as CyberGlove, have more than one contact points and output force feedback in parallel.

Typical haptic interaction concentrates on two issues: Haptic Collision Detection and Haptic Rendering. By considering time in discrete fashion, Haptic Collision Detection constantly checks if the virtual representation of haptic contact point collides with a certain substance in the virtual scene. The virtual representation of a contract point can be a point, a vector, or a 3D object. Haptic Rendering refers to generating the force feedback to the user depending on the position of the contact point with reference to the objects in the haptic scene. It conveys to the user additional information about the virtual environment synchronously with its visual rendering.

Haptic Collision Detection is similar but not identical to traditional Collision Detection in Computer Graphics. Besides the update rate difference (1000 Hz for haptic vs. 30 Hz for visual rendering), they also differs in the way how the virtual objects in the scene were defined. Although there have been efforts in performing collision detection with curved solid models [1], high density points [2], polyhedral models [3], superquadratic models [4], subdivision surfaces [5], and implicit functions [6], the most commonly used approach is still based on collisions with polygon meshes. There have been several popular haptic collision detection algorithms such as H-Collide [7], which adopts spatial subdivision and OBB trees, and Ruspini [8], which uses hierarchical bounding sphere for collision detection. Nevertheless, collisions with polygons suffer from a problem that they rely on a primitive level input such as vertices. For standalone applications, collecting primitives is fairly easy, however, for other applications, such as plug-ins to existing software tools, the primitive level Collision Detection algorithms are not feasible since the existing software may not provide APIs for retrieving primitives. This issue can be solved by using collision detection based on using implicit functions. For implicit functions, it is rather trivial to implement the collision with the contact point membership predicate. In [9], we proposed an approach to haptically collide with implicit functions, which does not need the primitive level input. In [6], another implicit function based approach was proposed for rendering large geometry models at 1000 Hz rate however it retrieves surface information from volumetric data for penalty-based force generation.

Haptic rendering mainly concerns two issues: position/orientation of the Contact Point and Contact Force that is sent back to the user. Naturally, as well as historically based on the existing applications, haptic rendering follows visual rendering pipeline. This influenced the most common way of haptic rendering with the visible surfaces which are typically displayed using polygons. Simple and straightforward, this approach suffers from the same problems as the polygon-based visualization–polygon
complexity. After a certain number of polygons participated in the scene, haptic rendering cannot be performed with the required 1000 Hz frequency. Since visual rendering requires as low as 30 Hz for refreshing images, this imposes two orders of magnitude more demands to the processing speed and reduces, respectively, the number of polygons which can be processed haptically. Compared to the maximum number of polygons that can provide for smooth visual interaction, the typical number of polygons providing for smooth haptic interaction is very much below it. Based on our laboratory test on a mid-level workstation, by using general polygon rendering approach and standard haptic interaction SDK, the fast update rate for smooth haptic rendering can only be achieved with no more than one million polygons. In comparison, the same workstation is able to smoothly render millions of polygons visually.

There are efforts in solving this problem from several perspectives, such as hierarchies of bounding volumes [10], spatial partitioning [11], and GPU-based acceleration [12]. Other approaches have also been studied, such as a velocity driven haptic rendering approach [13] which is based on LOD and render either coarse or detailed models depending on the velocity the user moves the haptic point.

Furthermore, things get further complicated if haptic rendering is performed not only with an elementary haptic object but with a more complex geometric object representing the virtual haptic tool. Collision detection between objects with or without physics will reduce the performance in terms of the participating polygons even further. In [14], the authors proposed a combination approach for accelerating object-object contact detection, such as geometric locality, temporal coherence and prediction. Besides haptic rendering with the visual surfaces, there are also a few works on haptic rendering of the interiors of solid objects and forces that can be directly used for calculation of force feedback. In [15], a voxel sampling approach has been proposed to accurately render complex real-world task. In [16], a solution was proposed to haptically render the volume data using dynamic spline-based implicit functions.

Currently there are several haptic SDK available for building a complete haptic interaction pipeline: CHAI3D, Reachin API&HaptX, OpenHaptic Toolkit and H3D&HAPI.

CHAI3D is an open source SDK that supports different haptic devices from different companies. It adopts OpenGL for visualization, ODE (Open Dynamics Engine) for rigid body collision detection, and GEL Dynamics Engine for simulating deformable objects.

Reachin API utilizes C++, Python and VRML to provide users with a flexibility of implementing haptic programs. HaptX is a haptics engine from Reachin, which works as a haptic toolbox for easy and fast development. It also works with different physics engines for interaction.

Open Haptic Toolkit from Sensable Company encapsulates functionalities for High-Level Collision Detection (HL API), which provides useful functions to directly handle geometric primitives and transformations. On the other hand, it constrained the scope of usage (only useful when the developer knows the polygons). In Open Haptic Toolkit, they also provide low-level control of the haptic device (HD API), which provides better flexibility to the developer. However, when using HD API, the developer has to design his own Collision Detection algorithm. Even though the Open Haptic Toolkit is very good, it only supports Sensable devices—other vendors and users have to develop their own SDKs.

H3D is a scene graph based API by SenseGraphics Company, which directly loads and parses X3D files, and then renders the scene both graphically and haptically. HAPI is developed by the same team and is intended to be a haptic rendering engine. It is device independent and purely built with C++. It utilizes OpenGL/DirectX for graphical rendering and OpenHaptic Toolkit/CHAI3D/God Object/Ruspini for haptic collision detection and force rendering.

II. ANALYSIS OF HAPTIC SDKS AND APPLICATIONS

The current haptic SDKs and applications have several drawbacks which need to be solved.

First, various haptic sensations such as feeling of surface tension and friction, as well as solid object viscosity and force fields, can only be provided separately and often at predefined, known to the haptic algorithm, locations of the haptic tools, while concurrent haptic rendering of these phenomena at any random location of the haptic tool is still a challenge. As an example of such efforts, CHAI3D supports multiple material effects such as viscosity, vibration and stiffness. However they implement stiffness as elasticity which prevents from penetrating the surface of the objects to enter the interior part to feel its viscosity.

Second, usually only one type of objects can be found in haptic scenes such as polygon-based surfaces, function-based surface models, or volumetric-based solid models. This simplifies the haptic interaction algorithms but greatly limits the abilities of the content creators.

Last but not least, commonly haptic scenes are simplified and tuned to perfect demonstrations. For example, the sizes of the haptic objects in a scene are always of the same order of magnitude so that the case of concurrent haptic rendering of very large and very small objects is simply excluded. Also, the haptic workspace is usually defined with a rigid size and location without abilities of scaling it up or down or moving it forward or backward for reaching objects with different sizes and locations. The initial position of the haptic handle is usually placed outside the objects in the scene, which exclude the situation of being initially inside objects with viscosity or force filed defined. Besides, haptic SDKs seldom consider inclusions or intersections of objects with different haptic properties. For example, in CHAI3D, when a small sphere is located completely inside a big sphere, properties from both spheres will be rendered when the handle is inside the small sphere, which may be explained as
reasonable but could be unrealistic for certain applications, where only properties from the small sphere need to be rendered.

In this paper we address the above problems and seek to propose a uniform approach and a generic framework for haptic interaction with objects with various haptic properties, different geometric models as well as different sizes and locations. We discuss the challenges and propose our solutions to the problems in Section III, and give implementation details in Section IV. In Section V, we show a few practical examples to illustrate our approach. We also critically analyze the results and outline future improvement plans. Finally, in Section VI we summarize the work done.

III. A UNIFORM ALGORITHM TO FORCE RENDERING IN COMPLEX HAPTIC SCENES

We aim to define surface and solid haptic effects as well as ubiquitous forces in the shared virtual scenes with mixed geometric models, including polygon meshes, point clouds, image billboards and layered textures, voxel models and functions-based models of surfaces and solids. We also propose a way how to identify location of the haptic tool in such haptic scenes as well as consistently and seamlessly determine haptic effects when the haptic tool moves within the scenes with objects having different sizes, locations, and mutual penetrations.

To introduce the haptic effects to the mixed geometric models, we propose that each haptic effect must have a certain geometric container within which this effect can be haptically rendered. Such a container may be a geometric surface of the object which is augmented with the haptic property (Fig. 1a). It can also be an invisible surface specially defined for the haptic effect and not necessarily coinciding with the geometric surface of the object (Fig. 1b and 1c).

![Figure 1. Haptic containers (dashed line): (a) Actual surface of the object, (b) Surface containing haptic forces, (c) Surface defining an object rendered without showing the actual surface (billboards, layered textures, point clouds).](image)

This approach allows us to create haptic effects for objects which do not have any surface at all such as point clouds or objects displayed as layered textures (e.g. MRI images). This approach also allows us to add ubiquitous forces to the scenes by encapsulating them into the invisible haptic containers. For each of the haptic container, be it a real surface of the object or a specially defined surface for the haptic effect, we allow for concurrent definition of surface properties (tension and friction), solid properties (density), and a force field.

To be able to consistently and seamlessly switch between the surface and inner solid and force properties, we revise the concept of surface stiffness, commonly used in haptic SDKs, which defines how fast the force vector will increase as the haptic tool penetrates beneath the object surface. The result of such surface force generation is unrealistic since the user always feels the surface (Fig. 2a) rather than the inner viscosity when both properties are defined. On the other hand, when viscosity is defined, no surface rendering can be performed (Fig. 2b). To provide for haptic rendering of both surface and inner haptic effects, we propose to introduce a certain surface zone within which the surface tension and friction effects will be activated while still allowing the haptic tool to penetrate the surface and render the forces resulting from the inner haptic effects (Fig. 2c). The depth of the surface zone may depend on the precision, workspace and maximum force of haptic device.

![Figure 2. Common surface (a) and viscosity (b) rendering, and (c) combined surface, viscosity and force field rendering with the surface zone defined.](image)

In contrast to the commonly used predefined location of the haptic tool and prior knowledge of the haptic effects to be rendered, we propose an algorithm identifying the location of the endpoint of the haptic tool—Haptic Interface Point (HIP)—in the shared virtual scene. Our approach is

![Figure 3. Casting haptic rays to detect the virtual tool location: (a) the tool is inside a haptic object, (b) the tool is close to several objects and it is also inside another larger container. The inner haptic property of the larger container will be rendered.](image)
based on casting multiple haptic rays in different directions from the current position of the HIP as soon as the tool arrives in the scene (Fig. 3). The rays are cast to find the nearest haptic container within which the HIP is located. The respective haptic effect will be then rendered. The number and the length of the rays can be adaptively adjusted by the local complexity of the scene, as well as manually by the user.

When the tool moves through the scene, in contrast to the common approach of colliding with polygons using one ray following the direction of the haptic stylus [17], we suggest a continuous multiple-ray casting to switch between the three types of haptic effects (surface, solid, and force field). Compared with [17] which requires a pre-computation phase to build hierarchical database of the scene, our approach does not require such prior knowledge of the scene, while it could still immediately identify the HIP location and start force rendering on the fly. We also propose a stack-based algorithm to permit mutual penetrations of the objects with different haptic effects defined (Fig. 4). As the HIP moves through various haptic containers, their identifiers are pushed into the stack. Each time the HIP leaves a container, the respective entry is popped out from the stack and the haptic effect corresponding to the next container stored at the top of the stack is rendered.

To be able to reach different parts of the scene with a different haptic precision, we introduce a movable and resizable 3D haptic zone which maps to the actual workplace of the haptic device. This approach corresponds to an expandable virtual hand which size can change as well (Fig. 5).

IV. ALGORITHM DETAILS

The algorithm is based on a few atomic operations:
- Multiple haptic rays casting;
- Haptic rays intersections with haptic containers;
- HIP position location;
- Surface properties rendering; and
- Inner properties rendering.

Multiple haptic rays casting is based on using parametric functions in spherical coordinates $u$, $v$ and $R$:

$$x = R \cdot \cos(u)$$
$$y = R \cdot \sin(u) \cdot \cos(v)$$
$$z = R \cdot \sin(u) \cdot \sin(v)$$

A uniform sampling of $u$ and $v$ provides for an even distribution of the rays in the scene around HIP to the distance $R$ (Fig. 6).

Each haptic ray cast is tested for possible intersections with haptic containers. The containers can be represented either by commonly used polygon mesh based surfaces or defined by mathematic functions (parametric and implicit). In case of polygon meshes, a trivial intersection of a parametrically defined ray with the polygons is performed. It can be accelerated by many commonly used ways. In case of parametrically defined surfaces of the containers, the polygon mesh is first calculated and then intersected with the rays. In case of implicitly defined surfaces, the haptic ray intersection can be performed at the level of function definitions. This procedure is similar to that of ray tracing and is performed by iterative binary subdivision steps which eventually result with the coordinates of the intersection point on the surface of the container. Alternatively, the implicitly defined surface can be also polygonized first however the direct evaluation method has a higher precision, requires less computations and hence provides for a higher performance.

HIP position location assumes understanding of whether the HIP is inside or outside the haptic container, and if inside,
how far from the surface it is located. In case of polygonal representation of the containers, the inside/outside predicate is implemented as a dot product
\[ g = R \cdot N \]
of a haptic ray \( R \) and a normal vector \( N \) to the surface at the intersection point (Fig. 7a). The HIP is inside the container if \( g > 0 \) and outside if \( g < 0 \). The distance to the surface then is approximated as a minimum of distances to polygon planes for the rays cast \( D = \|R - N\|/\|N\| \) (Fig. 7b).

![Figure 7. HIP location and distance calculation for polygon-based haptic containers](image)

In case of implicitly defined surfaces of the container, the inside/outside predicate is evaluated by analysis of the surface defining function sign which changes on different sides of the surface. Hence, using function definitions \( g = f(x,y,z) \geq 0 \) for the containers, the HIP is inside if \( g > 0 \) and outside if \( g < 0 \). Calculating the distance between the HIP and the implicitly defined surfaces we use a gradient to approximate the normal. In order to increase the accuracy of the normal, we use both positive and negative delta and then calculate the average:

\[
\begin{align*}
X_{\text{normal+}} &= (F(x + \Delta X, y, z) - F(x, y, z)) / \Delta X \\
Y_{\text{normal+}} &= (F(x, y + \Delta Y, z) - F(x, y, z)) / \Delta Y \\
Z_{\text{normal+}} &= (F(x, y, z + \Delta Z) - F(x, y, z)) / \Delta Z \\
X_{\text{normal-}} &= (F(x - \Delta X, y, z) - F(x, y, z)) / \Delta X \\
Y_{\text{normal-}} &= (F(x, y - \Delta Y, z) - F(x, y, z)) / \Delta Y \\
Z_{\text{normal-}} &= (F(x, y, z - \Delta Z) - F(x, y, z)) / \Delta Z
\end{align*}
\]

\[ N_X = (X_{\text{normal+}} + X_{\text{normal-}}) / 2 \\
N_Y = (Y_{\text{normal+}} + Y_{\text{normal-}}) / 2 \\
N_Z = (Z_{\text{normal+}} + Z_{\text{normal-}}) / 2 \]

The distance is then calculated using the same formula as for the polygonal representation.

Surface properties rendering activates when the HIP is inside of the container within its surface zone. Forces such as tension and friction generate in the direction opposite to the HIP movement. In case of the surface zone belonging to a polygonal container, the tension force is rendered as
\[ F = T_{\text{clipped}} * S_{\text{device_max}} * D / L_{\text{SurfaceZone}} \]
where \( T_{\text{clipped}} \) is the clipped tension value of the HIP, \( S_{\text{device_max}} \) is the maximum output force of the current haptic device, \( D \) is the distance between the HIP and the container surface, and \( L_{\text{SurfaceZone}} \) is the depth of the surface zone at the current position. Clipping is introduced here to cut off the big value and scale it to the range which the haptic device can render.

In case of the surface zone belonging to an implicitly defined container, the tension force is rendered as
\[ F_{\text{Tension}} = N_{\text{Approx}} * S_{\text{device_max}} * T_{\text{clipped}} - V_{\text{interpolated}} \]
where \( N_{\text{Approx}} \) is the approximated normal, \( S_{\text{device_max}} \) is the maximum stiffness which the haptic device could provide, \( T_{\text{clipped}} \) is the clipped tension value of the HIP, and \( V_{\text{interpolated}} \) is the interpolated velocity. The velocity is interpolated to neutralize the variations of abrupt force direction change due to approximated normal calculation.

The surface friction is calculated after tension calculation:
\[ F_{\text{Friction}} = - \text{Friction}_{\text{current_pos}} * F_{\text{Tension}} * V_{\text{interpolated}} \]
where \( F_{\text{Tension}} \) is the magnitude of the tension force and \( \text{Friction}_{\text{current_pos}} \) is the friction value of the HIP.

Inner properties rendering activates when the HIP is inside the haptic container but outside the surface zone. For density rendering, the force will generate in the direction opposite to the HIP movement and proportional to the density value. We obtain the velocity of the HIP at the haptic frequency and interpolate it as follows:
\[ V_{\text{current interpolated}} = C_1 * V_{\text{previous}} + C_2 * V_{\text{current}} \]
where \( V_{\text{previous}} \) is the velocity at 1/1000 sec before current haptic frame and \( C_1 \) and \( C_2 \) are interpolation coefficients.

The feedback force is then calculated as follows:
\[ F = V_{\text{current interpolated}} \cdot L \]
where \( L \) is the function-defined density evaluated at the HIP.

When a force field within a haptic container is defined, the forces there define the direction of the HIP movement (Fig. 8a). Since the haptic pipeline executes at 1000Hz, the HIP will be continuously guided to move from one point to another (Fig. 8b).

![Figure 8. Changes of movement direction between haptic frames generate smooth HIP moving path](image)

The algorithm workflow is shown in Figure 9.
V. TECHNICAL DETAILS AND APPLICATION EXAMPLE

To implement the proposed approach, we need a generally accepted language platform which requires no direct input of polygons from the user and provides common interfaces for non-specific haptic devices. BS Contact from Bitmanagement is widely used for building virtual environments and it supports VRML, X3D and COLLADA. Besides, it provides standard COM (Component Object Model) interfaces for supporting various input devices including haptic devices. BS Contact also has some auxiliary methods for interactions, such as ComputeRayHit, which are quite helpful for certain implementations. Therefore, we selected VRML/X3D and BS Contact as our test bed.

To be able to support devices from different vendors, we abstracted a group of interfaces that a single-point desktop haptic device could commonly utilize into an “Abstract Device”. For different devices from different vendors, the “Abstract Device” maps their specific function calls into the common function calls. If some of the device-specific functions are missing (i.e. some devices may only have 3DOF input), the developer will be able to either ignore these missing input, or design an alternative way to implement them. By using this approach, we have successfully implemented rotation functions on the 3DOF input Novint Falcon by assigning 3 device buttons as roll, pitch yaw rotations, respectively, and therefore converted it into a 6DOF input device.

Our software is implemented as a plug-in to BS Contact VRML/X3D browser which works with MS Internet Explorer and Mozilla Firefox. A specially crafted string in the url field of the Script node of this browser allows it to automatically load and execute functions from a dynamic link library file. The library is written in C++, which generates native codes running on the machine. The parser module of the plug-in is implemented using Yet Another Compiler Compiler (YACC) for C. The input file of YACC is a text file which contains the grammar definition of the programming language which is written in a form similar to Backus Naur Form (BNF). By running the YACC executable, the input file is converted into a C/C++ source file, which parses the grammar of the function definitions.

An EXTERNPROTO is declared in VRML/X3D to identify names and data types of the interfaces, which will be responsible for retrieving hardware information from and send back data to the “Abstract Device”.

An example of using the uniform approach for mixed haptic rendering is illustrated in Fig. 10. In this orthopedic surgery training application, the user is supposed to use a haptic device to insert a needle through the bone. Various haptic effects such as tension, friction and density are defined with mixed geometric models as containers. Also, the objects mutually penetrate.

The bone is a polygonal model reconstructed from MRI data. The skin and bone marrow geometries are defined by implicit functions. The skin is also defined with a medium friction and low tension to simulate skin. Under the skin, high viscosity is defined to simulate the muscles. The bone surface is defined with a medium friction and very high tension, and the marrow layer has no surface effect but just low viscosity inside.
In Fig. 11 we illustrate it with a simplified code.

As the user moves the HIP to penetrate the objects, depending on the magnitude of the force he exerted on the haptic device, the user will feel the skin, the muscles (Fig. 11a), the bone and the marrow (Fig. 11b) with different tension, friction and viscosity. The object stack is also used to pop-up previously touched objects as the HIP out of the objects. Mixed object definitions as well as various haptic effects are totally transparent to the user.

VI. Conclusion

We proposed a uniform approach to define surface and solid haptic effects as well as ubiquitous forces in the shared virtual scenes with mixed geometric models, including polygons meshes, point clouds, image-based billboards and layered textures, voxel models and functions-based models of surfaces and solids. We also proposed a way how to identify location of the haptic tool in such haptic scenes as well as consistently and seamlessly determine haptic effects when the haptic tool moves within the scenes with objects having different sizes, locations, and mutual penetrations.

We implemented a haptic plug-in to Bitmanagement BS contact VRML/X3D browser. We described the architecture of the plug-in and the branches of the core collision detection algorithm. Various haptic effects can be concurrently rendered by touching geometric container of the objects, which may or may not coincide with the actual geometric surface. Still staying within the VRML/X3D rendering pipeline, we are able to perform concurrent haptic rendering on VRML and X3D scenes mixed defined by functions and polygons, with different desktop haptic devices from different vendors.

This uniform approach has been adopted for several standalone and collaborative applications which have been published in [18, 19, 20]. Videos illustrating the results of our study can be seen at http://www3.ntu.edu.sg/home/assourin/fvrm/video.htm and http://www3.ntu.edu.sg/home/assourin/fvrm/video1.htm.
ACKNOWLEDGMENTS

This project is supported by the Singapore National Research Foundation Interactive Digital Media R&D Program, under research Grant NRF2008IDM-IDM004-002 “Visual and Haptic Rendering in Co-Space”.

REFERENCES


