Realistic Cross-Platform Haptic Applications Using Freely-Available Libraries

Cristian Luciano
Pat Banerjee
Thomas DeFanti
2039 ERF (M/C 251), 842 W. Taylor St.
University of Illinois at Chicago
Chicago, IL 60607
clucia1@uic.edu
banerjee@uic.edu
tom@uic.edu

Sanjay Mehrotra
Northwestern University
Evanston, IL
mehrotra@iems.nwu.edu

Abstract

This paper describes the development of a generic framework for implementing realistic cross-platform haptic virtual reality applications. Currently, freely available Software Development Kits (SDKs) deal with a single Haptic Interaction Point (HIP), i.e. the tip of the haptic probe. However, many applications as path planning, virtual assembly, medical or dental simulations, as well as scientific exploration require object-object interactions, meaning any part of the complex 3D object attached to the probe collides with the other objects in the virtual scene. Collision detections and penetration depths between 3D objects must be quickly computed to generate forces to be displayed by the haptic device. In these circumstances, implementation of haptic applications is very challenging when the numbers, stiffness and/or complexity of objects in the scene are considerable, mainly because of high update rates needed to avoid instabilities of the system. The proposed framework meets this high requirement and provides a high-performance test bed for further research in algorithms for collision detection and generation of haptic forces.

1. Introduction

Haptic rendering techniques can be classified based on the way the probing object is modeled. It could be represented as:

- a point
- a line segment
- a complex 3D polyhedron

In *point-based* haptic interactions, only the end point of the haptic device interacts with the virtual objects. In each frame, the collision detection algorithms checks if the HIP is inside the virtual object. If so, penetration depth is computed as the distance between the current HIP and the corresponding surface contact point. This technique is extremely fast but is not capable of simulating tool-object interactions that involve single or multiple objects in contact with the tool at arbitrary locations of the tool [1][2][8][10][13][18].

When the probe is modeled as a set of line segments, collisions are checked between the finite lines and the objects. In this *ray-based* technique the geometric model of the probing object is simplified to a set of connected line segments simulating long tools. This method is not as fast as point-based but few computations are needed. Collision between many objects can be detected; however it is inappropriate if the probing object has a complex geometry which cannot be modeled using line segments [3][10].

The most computationally expensive, but most realistic, technique considers the probing object as a *complex 3D polyhedron*, and so collisions between vertices of its geometry and the remaining objects in the scene are checked every time the user moves the probe of the haptic device [9][11].

Object-object collision detection could be enormously time consuming for very complex objects. However, relatively complex 3D objects can be decomposed into multiple convex subparts during an off-line pre-processing [4][16]. Since collision detection algorithms for convex objects are faster than those for arbitrary geometry objects, we assume a previous decomposition phase and we deal only with convex primitives.

The motivation of this work is to provide a framework to develop 3D-object-based haptic applications using non-commercially available SDKs. For that purpose, state-of-the-art freely-available cross-platform graphics and collision detection libraries are evaluated.

2. Haptics libraries

GHOST (General Haptic Open Software Toolkit) is the library provided by [15] along with its PHANToM devices. GHOST is a C++ object-oriented toolkit that enables application developers to interact with haptic

devices and create haptic environments at the object or effects level. Using the GHOST SDK, developers can specify object geometry (similarly to OpenGL) and properties, or global haptic effects, using a haptic scene graph.

GHOST provide functions to automatically compute the interaction forces between the HIP and objects or effects within the scene, and send forces to the haptic interaction device for display. The abstraction of classes, methods, manipulators, and effects facilitates the creation of very simple haptic applications.

An alternative C++ Application Programming Interface (API) is e-Touch, developed by [12] to allow rapid development of haptic applications using OpenGL for graphic rendering. Basically, e-Touch can be thought as a wrap around GHOST to allow an Open Module community of researchers to work together in a common development environment and easily develop haptic applications. The source code is, then, freely available to researchers. A point-based technique is used by e-Touch to model the haptic probe as well.

3. Graphics libraries

GHOST and e-Touch functions available to generate 3D objects and to manipulate the virtual scene are very limited,. The capability of loading and showing VRML files is non-existent. Open Inventor represents a much more sophisticated API than OpenGL. This object-orientated cross-platform graphics library is based on OpenGL, but provides a plethora of advanced tools for both manipulation and visualization of the virtual environment. Open Inventor has become the de facto standard graphics library for 3D visualization and visual simulation software in the scientific and engineering community [5]. It is an object-oriented toolkit that simplifies and abstracts the task of writing graphics programming into a set of easy to use objects. A variety of fundamental application tasks such as rendering, picking, event handling, and file reading/writing are built-in operations for all objects in the database and are thus simple to invoke. It is quite easy to use and it makes 3D direct manipulation programming possible.

4. Collision detection libraries

Collision detection is the bottleneck of most haptic applications, mainly because of the extremely high computational requirements. Haptic update rates must be on the order of 1 kHz to maintain a stable system. Although collision detection has been studied in computer graphics for many years, not all existing algorithms are powerful enough to meet the real-time requirements of haptic rendering. We have evaluated some algorithms which are freely available.

According to our experiments, SWIFT++ (Speedy Walking via Improved Feature Testing) [6] is capable of detecting collisions of convex objects in expected constant time, but it is not robust enough. We have tested it with random object movements and found cases in which it fails to detect collisions.

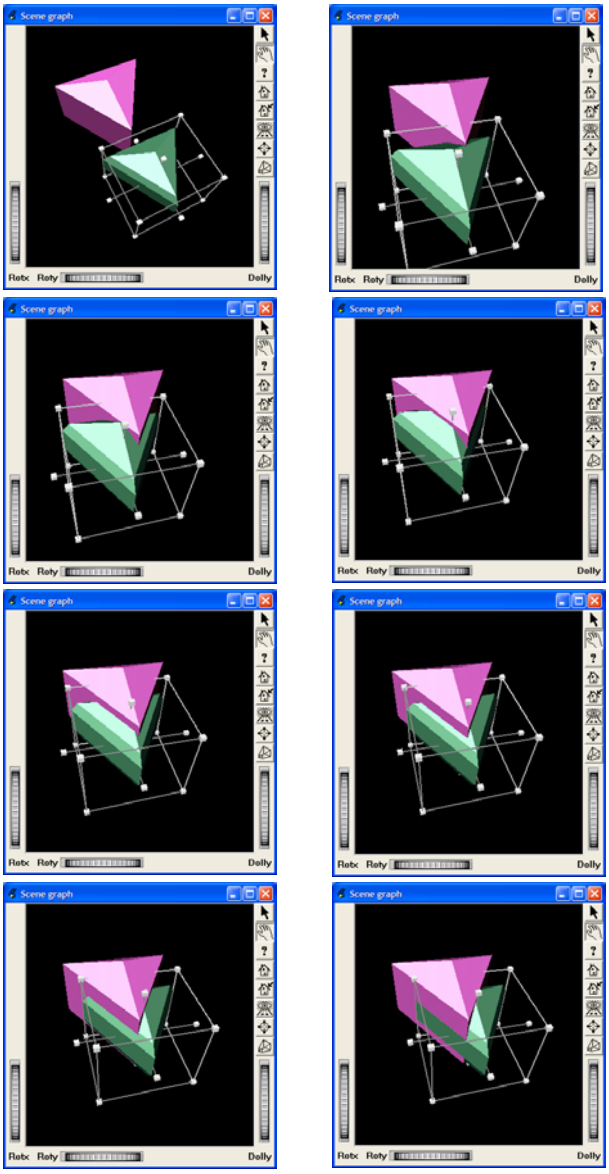

Figure 1: Situation in which SWIFT fails to detect collision

Figure 1 shows a sequence of movement between two identical convex objects in which SWIFT does not detect the collision. We believe that this may be caused by inconsistencies of the lookup table used by SWIFT++, or in the heuristics used. However, further research should be conducted to find out the real cause, which is beyond this project. Failing to detect collision is the worst problem in the application of these algorithms in haptics,

since the object attached to the probe can penetrate other objects in the scene, without obtaining any force feedback.

In addition to this, SWIFT++ provides the set of closest features (vertex, edge, face) for a pair of disjoint convex objects. However, when an intersection is determined, it neither measures nor estimates penetration depth, which is very an useful piece of information to compute reaction forces to be sent to the device in response to the probe position and orientation. Therefore, the penetration depth of the current frame should be estimated according to the previous frame, which may be not accurate when objects were interpenetrating over a number of frames.

DEEP (Dual-space Expansion for Estimating Penetration Depth) [11] is an incremental algorithm to estimate the penetration depth between convex polytopes. DEEP is an extension of SWIFT++ since it is designed and implemented on top of it. Unfortunately the freely available source code presents implementation problems and it crashes frequently even with very simple convex polyhedral. Moreover, since it depends on SWIFT to detect collisions, it is not reliable.

SOLID v 3.1 (Software Library for Interference Detection) [17] is a freely available pre-compiled library for many platforms (the source code is commercially available, though). SOLID is fast, robust and very useful for detecting collisions and computing penetration depth between pairs of convex objects that are moving relative to one another over time.

5. Proposed implementation

We have carried out several experiments with GHOST and e-Touch, as well as the freely available collision detection libraries to determine their capabilities and deficiencies for this project. Since a point-based technique is implemented by both GHOST and e-Touch, neither of these libraries is appropriate for creating complete object-based haptic applications. However, the latest version of GHOST provides low-level access to the PHANToM device. This enhancement enables users to obtain raw encoder counts, send forces directly to the haptic device and receive motor temperatures without using the higher-level GHOST scene graph, object primitives, etc.. Consequently, we decided to restrict GHOST to perform the actual communication with the force feedback device.

Specifying the object geometry directly in OpenGL and manipulating the virtual scene with the limited graphics functions provided by GHOST or e-Touch are cumbersome for complex objects and user interactions with the virtual environment. Open Inventor provides outstanding high-level functions for manipulation with the mouse, or trackball, but it does not support haptic devices. Hence, for visualization and interaction with the scene, we propose to extend the Open Inventor API to support the PHANToM device and provide the user with a more efficient tool than the primitive OpenGL. Now 3D objects can be drawn using any 3D design software like 3DStudioMax, Maya, Pro/E, etc. and then converted to Open Inventor format. Once those files are loaded by the application, the user can manipulate the virtual environment and feel the objects with the haptic probe.

The robustness problems with SWIFT and the unstable implementation of DEEP make SOLID the best currently existing option for object-based haptic applications and, therefore, it was used for the implementation of this project.

6. Haptics and graphics rendering

As we have seen, the scene graph is maintained by Open Inventor, which allows us to handle the camera, the objects in the virtual environment and the object attached to the virtual probe (see Figure 2).

While haptic update rates needs to be approximately 1,000 Hz (otherwise, virtual surfaces feel softer or the haptic device vibrates), update rates of 30 Hz are fast enough for graphics rendering. We have designed the Phantom Transform as a new class derived from the Open Inventor Transform class. Basically, the Phantom Transform class communicates with the device retrieving the current probe position and orientation, and sending forces to be applied by the haptic device (in one thread), while Open Inventor renders the scene graph (in the other thread). Consequently, the haptic and graphic upgrades are performed by two independent threads, which can be executed concurrently in a dual-processor system.

7. User interaction modes

We have designed two user interaction modes. In the *selection* mode, the user can select a particular object in the virtual environment touching it with the 3D cursor, modeled as a three dimensional point and shown as a small sphere (see Figure 3). A similar point-based haptic technique is then used to detect the collision between the 3D cursor and the objects in the scene. Once the 3D cursor is inside a particular object, the user can feel as the tip of the probe penetrates the object, which is shown as "selected" by the Open Inventor Selection node (see Figure 4). After pressing the button of the haptic device stylus, the selected object is then attached to the probe (so the 3D-Cursor Separator is replaced by the selected Object Separator node, which becomes a child of the Phantom Separator node).

Now in the *grabbing* mode (see Figure 5), a 3D-polyhedron-based technique is implemented to detect the collision between the selected object and the remaining objects in the virtual scene. Contacts are felt according to different object materials that define different force feedback sensations. We will see how to compute the forces to be applied by the haptic device in following sections.

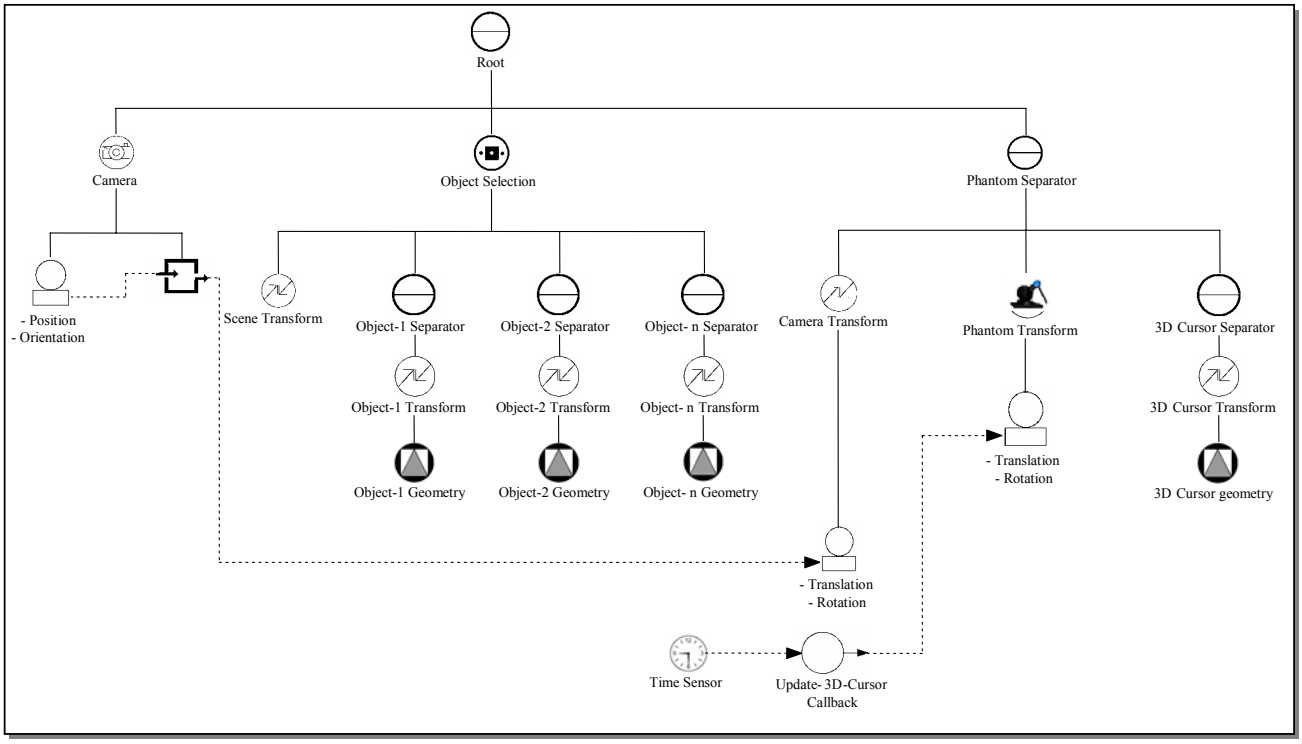

Figure 2: Scene graph

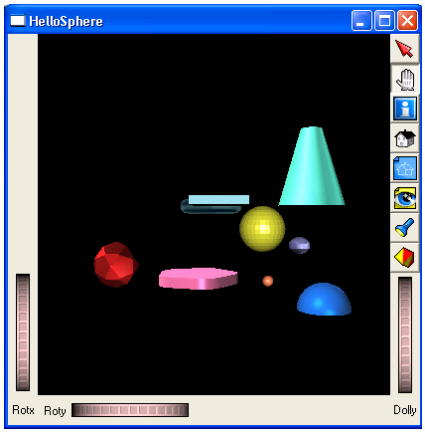

Figure 3: 3D objects and cursor

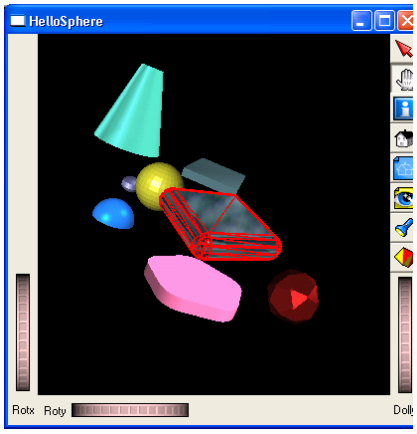

Figure 4: Selection mode

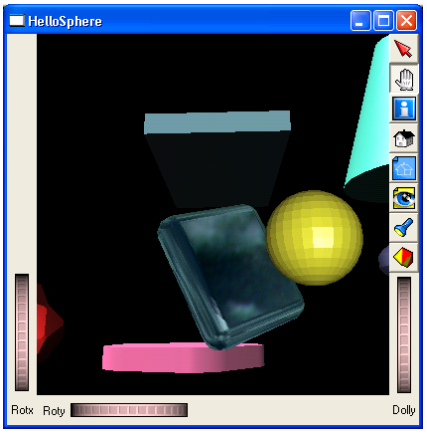

Figure 5: Grabbing mode

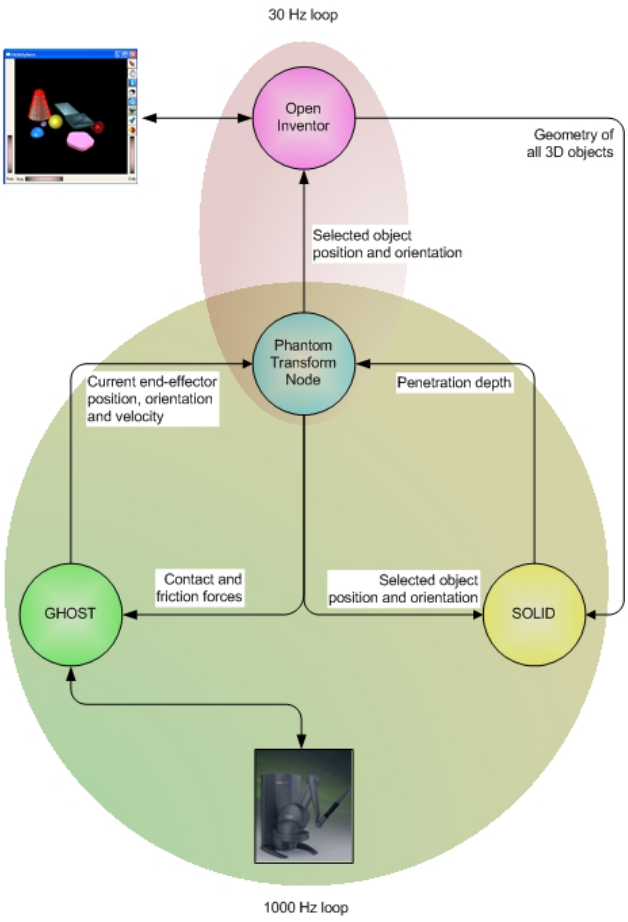

Figure 6: Phantom Transform node

8. Phantom Transform node

The interaction with the haptic device is performed through the GHOST gstDeviceIO class [15]. This low-level class allows us to define a callback function which is called automatically 1,000 times per second, creating a 1 kHz servo loop running in a separated thread. At the same time, the scene graph is traversed (rendered) by Open Inventor 30 times per second. shows how the specialized Phantom Transform node creates the connection among Open Inventor, GHOST and SOLID, encapsulating the following tasks:

- Initialize the PHANToM device and enable forces
- Read the encoders to get position, orientation and velocity of the haptic end effector
- Transform position and orientation of the object attached to the probe from the camera coordinate system to the world coordinate system, and update the SOLID library
- Ask SOLID to check the collision between the virtual probe and all the objects in the scene
- Check the status of the PHANToM switch to toggle between selection and grabbing modes
- Compute and send the forces to the haptic device.

9. Computation of forces

Since GHOST has no information about the geometry of objects in the scene, we need to generate the forces to be sent to the PHANToM device manually. Each object in the virtual scene may have a different material, which is defined by four parameters: stiffness S, viscosity V, stickiness T and friction F coefficients (based on [14]).

When a contact is made, SOLID gives two collision witness points for each pair of colliding objects P_0 and P_1. In order to compute the contact forces, surface normals must be calculated. Given the camera transformation matrix CM, the normal vector N is defined as:

$$N = (P_1 - P_0) * CM$$

Based on Hooke's law, contact forces Cf are calculated as:

$$Cf = N * MaxStiffness * S_0 * S_1$$

where $MaxStiffness$ is a parameter given by GHOST according to the model of Phantom device that is being used.

Adding viscous damping to the equation enhances the user's perception of hard surfaces. Viscosity forces Vf are inversely proportional to the end-effector velocity when the probe is colliding with an object. Since gstDeviceIO also provides velocity information, they can be modeled as:

$$Vf = Velocity * V_0 * V_1$$

According to Coulomb model, friction applies a retarding force that is a function of a coefficient of friction and tangential forces in the direction opposite of the direction of motion. To implement friction, just after a collision, the contact points, P_0 and P_1, of both colliding objects are stored. If the user slides one object along the surface of the other, tangential forces to the surfaces of both colliding objects must be computed to restore the user back to the initial contact points. Then, if the sliding force applied by the user exceeds certain threshold, new contact points are stored and this process starts again.

The witness points are given with respect to the world coordinate system. But they must be expressed relative to both objects, so they are transformed to each object

coordinate system. Given the first P_F and the current P_C contact points after sliding the probe, and two coefficients F_0 and F_1 that define material friction; we compute respective tangential forces for each colliding object as:

$$Tf_0 = \left(P_{C_0} - P_{F_0}\right) * F_0$$
$$Tf_1 = \left(P_{F_1} - P_{C_1}\right) * F_1$$

If the magnitude of Tf_i is greater than the stickiness coefficient T_i, P_{C_i} becomes the new P_{F_i} and Tf_i is ignored. Otherwise, it contributes to the friction force Tf, which is defined as:

.

$$Tf = Tf_0 + Tf_1$$

The total force for each pair j of colliding object is computed as:

$$Force_j = C_f - V_f + Tf$$

The displayed force is a function of the j forces computed for each pair of colliding object. Experiments have demonstrated than smoother force effects are obtained when the final force to be sent to the device is calculated as:

$$FinalForce = \frac{\sum_{k=1}^{j} Force_j}{j}$$

10. Conclusions and future research

This paper shows a successful approach to integrate Open Inventor, SOLID and GHOST for the creation of realistic haptic applications. We have demonstrated that it is possible to obtain excellent results merging the best of three worlds: an advanced and sophisticated graphics library, a fast and reliable collision detection algorithm, together with a library to get direct access to the PHANToM device. A simple but very efficient way to model multiple contact and friction forces between 3D objects to be displayed by a 3-DOF haptic device is also shown. The main contribution of this work is the development of a cross-platform and modular test bed for further research and implementation of new collision detection algorithms as well as new contact and friction models for haptics.

We plan to extend our current system to allow more complex non-convex objects, stereo visualization of the virtual scene for semi-immersive virtual reality environments as well as haptic texture rendering. This enhancement would require more computation power to meet the high real-time requirements imposed by haptics. Parallel processing may be a reasonable solution.

Haptic telecollaboration is another challenging area of research, mainly because of network issues such as delay, jitter (variation of delay), reliability or bandwidth which may cause severe deterioration of the performance of the system. In networked remote haptic interaction, delay may cause not only time lag between human operation and force feedback, but also instability of the haptic device. These problems make haptic telecollaboration prohibitive or at least very hard to implement with regular networks. However, the use of state-of-the-art clusters of low-cost PCs interconnected by low latency, high bandwidth and high speed optical networks may allow thinking on high performance haptic telecollaboration as a feasible task.

11. Acknowledgments

This research was supported by NSF grant DMI 9988136, NIST ATP cooperative agreement 70NANB1H3014, Fulbright/YPF-Repsol scholarship program, and the Department of Mechanical and Industrial Engineering and the Department of Periodontics at the University of Illinois at Chicago (UIC).

Additional support was obtained by the virtual reality and advanced networking research, collaborations, and outreach programs at the Electronic Visualization Laboratory (EVL) at the UIC, which were made possible by major funding from NSF awards EIA-9802090, EIA-0115809, ANI-9980480, ANI-0229642, ANI-9730202, ANI-0123399, ANI-0129527 and EAR-0218918, as well as the NSF Information Technology Research (ITR) cooperative agreement (ANI-0225642) to the University of California San Diego (UCSD) for "The OptIPuter" and the NSF Partnerships for Advanced Computational Infrastructure (PACI) cooperative agreement (ACI-9619019) to the National Computational Science Alliance. EVL also receives funding from the US Department of Energy (DOE) ASCI VIEWS program. In addition, EVL receives funding from the State of Illinois, Microsoft Research, General Motors Research, and Pacific Interface on behalf of NTT Optical Network Systems Laboratory in Japan.

12. References

[1] Adachi, Y., Kumano T. and Ogino, K. (1995) “Intermediate representation for stiff virtual objects”. *Proceedings of the IEEE Virtual Reality Annual International Symposium* (pp. 203-210). Research Triangle Park., NC: IEEE.

[2] Avila, R. and Sobierajsky, L. (1996) "A haptic interaction method for volume visualization", *IEEE Proceedings of Visualization* (pp. 197-204).

[3] Basdogan, C., Ho, C. and Srinivasan, M. (1997) "A ray-based haptic rendering technique for displaying shape and texture of 3D objects in virtual environments. *ASME Winter Annual Meeting* (pp. 61 – 77-84). Dallas, TX: ASME.

[4] Chazelle, B. and Palios, L. (1997) Decomposing the Boundary of a Nonconvex Polyhedron. *Algorithmica* 17, 245-265

[5] Coin3D (2002) "The Coin Source", *www.coin3d.org*.

[6] Ehmann, S. and Lin, M. (2000) "Accelerated proximity queries between convex polyhedra by multi-level Voronoi marching", *Technical report*, Department of Computer Science, University of North Carolina.

[7] Ehmann, S. and Lin, M. (2001) "Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition", *Eurographics*, 20,(3.)

[8] Gregory, A., Lin, M., Gottschalk, S. and Taylor, R. (1999) "H-collide: A framework for fast and accurate collision detection for haptic interaction", *Proceedings of Virtual Reality Conference 1999*.

[9] Gregory, A., Mascarenhas, A., Ehmann, S., Lin, M. and Manocha, D. (2000) "Six Degree-of-Freedom Haptic Display of Polygonal Models", *Proceedings of 2000 IEEE Visualization*.

[10] Ho, C., Basdogan, C. and Srinivasan, M. (1999) "An efficient haptic rendering technique for displaying 3D polyhedral objects and their surface details in virtual environments", *Presence: Teleoperators and Virtual Environments*, 8(5), 477-491.

[11] Kim, Y., Lin, M. and Manocha, D. (2002) "DEEP: Dual-space Expansion for Estimating Penetration depth between convex polytopes", *IEEE International Conference on Robotics and Automation*

[12] Novint Technologies, Inc. (2001) "Using and Programming the e-Touch™ API, A 3D Haptic Human/Computer Interface and Software Development API. *www.eTouch3D.org*.

[13] Ruspini, D., Kolarov, K. and Khatib, O. (1997) "The haptic display of complex graphical environments", *ACM (Proceedings of SIGGRAPH)*, 345-352.

[14] Salisbury, K., Brock, D., Massie, T., Swarup, N. and Zilles, C. (1995) "Haptic rendering: programming touch interaction with virtual objects", *1995 Symposium on Interactive 3D Graphics*, Monterey, CA.

[15] SensAble Technologies, Inc. (2002) "GHOST® SDK API Reference and Programmer's Guide, Version 4.0", *www.sensable.com*.

[16] Tesic, R. and Banerjee, P. (2001) "Exact Collision Detection for Virtual Manufacturing Simulator", *IIE Trans,* vol. 33 (1), 43-54.

[17] Van Den Bergen, G. (2002) "SOLID Collision Detection Library" User's guide. *www.solid.org*

[18] Zilles, C and Salisbury, J. (1995) "A constraint-based god-object method for haptic display", *IEEE International Conference on Intelligent Robots and System, Human Robot Interaction, and Co-operative Robots*, 3. 146-151.