Contributing to VRPN with a new server for haptic devices (ext. version)

Maria Cuevas-Rodriguez, Daniel Gonzalez-Toledo, Luis Molina-Tanco, Arcadio Reyes-Lecuona
University of Malaga – {mariacuevas, dgonzalezt, lmtanco, areyes}@uma.es

This article is an extended version of the poster paper: Cuevas-Rodriguez, M., Gonzalez-Toledo D., Molina-Tanco, L., Reyes-Lecuona A., 2015, November. "Contributing to VRPN with a new server for haptic devices". In Proceedings of the ACM symposium on Virtual reality software and technology. ACM.

Abstract

VRPN is a middleware to access Virtual Reality peripherals. VRPN standard distribution supports Geomagic® (formerly Phantom) haptic devices through the now superseded GHOST library. This paper presents VRPN OpenHaptics Server, a contribution to VRPN library that fully reimplements VRPN support of Geomagic Haptic Devices. The implementation is based on the OpenHaptics v3.0 HLAPI layer, which supports all Geomagic Haptic Devices. We present the architecture of the contributed server, a detailed description of the offered API and an analysis of its performance in a set of example scenarios.

CR Categories: I.3.7: Three-Dimensional Graphics and Realism —Virtual reality. H.5.2: User Interfaces — Haptic I/O

Keywords: VRPN, haptic devices, OpenHaptics, Phantom.

1 Introduction

The demanding nature of haptic rendering, requiring 1kHz rates to deliver simulation of realistic forces, has made researchers explore different approaches to alleviate computational load [Otaduy et al., 2013]. One strategy consists in distributing the computational load among different CPUs. In order to do so, it is necessary to define a protocol for transferring information among these computers. In Virtual Reality applications, VRPN (Virtual Reality Peripheral Network) is a popular middleware to communicate devices and distribute Virtual Reality (VR) applications [Taylor II et al., 2001]. In this paper we present a novel VRPN OpenHaptics Server to distribute the haptic and graphical rendering onto different machines over a network.

The VRPN OpenHaptics Server, as the name indicates, is a VRPN server based on the OpenHaptics SDK, used to manage Geomagic Haptic Devices (company who acquired Sensable Technology Inc.'s® in 2012). VRPN is typically used for interfacing with motion capture and tracker devices in a VR system. An example can be seen in [Suma et al., 2011], where a Flexible Action and Articulated Skeleton Toolkit (FAAST) is presented. FAAST incorporates a VRPN server for streaming user's skeleton joints over a network. Another platform that uses VRPN protocol to broadcast tracking data is described in [Freeman et al., 2010]. VRPN package to handle input data from all control peripheral such as joysticks, gamepads, buttons and a tracking system

Beyond the control of tracking devices, VRPN also supports other peripheral as button devices, analog input, sounds, haptic devices, etc. [Renard et al., 2010] presents an open source software platform called OpenViBE that includes a VRPN server module to send analog and buttons values to a VR application. Also, [Jacobson et al., 2005] presents a CaveUT which incorporates. VRPN interface is provided by most popular tracker devices, and it is supported by many graphic rendering engines for tracking. Another example is presented in [Mashbrun et al., 2005], where authors describes a real-time data-display system for a microscope, the 3D-Force Microscope, that uses a client-server model and all the network communications use VRPN protocol.

The VRPN standard distribution includes a server to handle Geomagic® force feedback haptic devices (formerly Phantom haptic device), called VRPN Phantom server. This server has been implemented using the Ghost library, which is no longer supported by the company. In [Cuevas-Rodriguez et al., 2013] a haptic device server is presented, based in Haptic Library Application Programmable Interface (HLAPI) from OpenHaptics 3.0, the currently Geomagic haptic device toolkit. However, this interface was limited and the implementation had performance issues.

This present work is a contribution to the VRPN standard distribution with a new server which supports OpenHaptics devices. We improve upon [Cuevas-Rodriguez et al. 2013] with a novel implementation which follows VRPN coding rules, solves its performance issues and widely extends its interface. Development has been done in coordination with the current VRPN curators. At the time of submission the contribution is being evaluated to become part of VRPN standard distribution.

The paper is structured as follows. Section 2 describes the technical background necessary to understand every technologies used in this work. The third section explains the device features and the advantages that the implemented server presents. The OpenHaptics Server implementation is described in-depth in section 4. Next, section 5, describes how to use the OpenHaptics Server from the client and from the server side. Section 6 presents a study to evaluate the performance of the server. Finally, the conclusions are given in section 7.

2 Technical background

The VRPN OpenHaptics Server presented here relies on two technologies: Geomagic Haptic Devices and the VRPN library.

Geomagic is a company that offers software and force-feedback haptic devices that enable users to touch objects in a 3D virtual environment. GHOST (General Haptic Open Software) was the first commercial available Application Programmable Interface (API) to manage these haptic devices, but is no longer supported. The GHOST API has been replaced by the OpenHaptics™ toolkit [Itkowitz et al., 2005], an OpenGL based library. OpenHaptics™ is organized in three layers. The lower level library, HDAPI (Haptic Device API), allows the user to render forces directly. The next layer, HLAPI (Haptic Library API), provides high-level mechanisms to perform the haptic rendering. Finally, the QuickHaptics micro API level is an even higher level API which enables fast creation of haptic scenes with a minimal amount of code. HLAPI is the chosen library to implement the Phantom server, as: (1) the final users of our server will be developers with no prior knowledge of haptics fundamentals, but still wanting to add haptic properties to the graphic scenes; (2) QuickHaptics library does not offer enough control to manage the haptic rendering properties; and (3) HLAPI automatically computes the haptic rendering based on geometric primitives and offers commands to set haptic properties to the virtual objects, as friction, stiffness, damping, viscosity, etc.

OpenHaptics v3.0 SDK supports every Geomagic haptic device: the Geomagic Touch (formerly the Sensable Phantom Omni), the Geomagic Touch X (formerly the Sensable Phantom Desktop)

and the Geomagic Phantom Premium haptic device. These desktop devices use motors to provide force feedback through a mechanical arm with a pen-shape end-effector called stylus. To use the device, the users hold the stylus as they would do with a pen. The motors exert forces on the users' hand to simulate interaction between the pen tip and a virtual object. The devices used to test the new VRPN OpenHaptics Server have been the Geomagic Touch and the Geomagic Touch X. Both are 6 degrees-of-freedom (DoF) devices which can measure the 3D spatial position (along the cartesian x, y and z axes) and the orientation (pitch, roll and yaw) of the handheld stylus, while exerting forces along the three cartesian DoFs.

VRPN is an open source C++ library aimed at supporting distribution and modularity of virtual reality applications. The library provides a set of servers that allow communication between applications and interaction devices (e.g. mouse, joystick). The provided servers have common features and functionalities which are abstracted out in 5 canonical classes. Each server derives from one or more of the canonical classes: Tracker, Button, Analog, Dial and Force Device. The standard distribution of VRPN 7.33 offers a device-specific class to communicate a VR application with a Geomagic haptic device, the *vrpn_Phantom* class. This class performs the haptic rendering providing force feedback to user through the Geomagic device and inherits from the following canonical classes: *vrpn_Tracker* (which reports device position, orientation and acceleration), *vrpn_Button* (which reports device buttons state) and *vrpn_ForceDevice* (which reports device applied force). Most of the VRPN standard distribution methods to manage the haptic device force feedback are implemented using the currently obsolete GHOST library. Only a subset of functions such as device initialization and shutdown, use the OpenHaptics HDAPI library. Cuevas-Rodriguez et al. [2013] developed a server based on OpenHaptics HLAPI. However this server did not follow VRPN coding rules and was therefore never contributed to the VRPN. The server also had a limited interface and performance issues. The server introduced here (1) widely extends the functionality of the implementation by [Cuevas-Rodriguez et al. 2013]; (2) avoids performance issues by streaming data only when required; and (3) closely follows VRPN coding rules with the goal of contributing to the VRPN standard distribution by updating VRPN support of haptic devices.

3 OpenHaptics Server: features

VRPN is a middleware that allows distributing applications and devices over a TCP/IP network. In the case of the haptic server, the system interacts locally with the haptic device according to haptic scene, delivering the data to clients through a TCP/IP network. In this section we review the advantages of this approach. In general, transforming the architecture of a VR application into a distributed one has several advantages. When the application includes haptic interaction, this can be especially advantageous as the rendering loop can be computationally very expensive. The haptic server allows us to run as separate applications the haptic render and the graphic render. This is advantageous for several reasons, as (1) The computing power required for the simulation can be distributed between different systems; (2) separating the haptic and graph loop reduces the complexity of both with a small cost; (3) we keep the flexibility of running both applications (haptic and graph renderers) in the same or in different machine; and (4) information on the haptic scene can be fed into auxiliary applications within the virtual reality system, such as, for example, monitoring and evaluation applications (Figure 1). Finally, from a developer's perspective, the OpenHaptics libraries can be complex, while the interface offered by *vrpn_OpenHaptics* is much simpler. The distributed architecture and the simpler interface greatly facilitate adding haptic rendering to an existing VR system that does not support it.

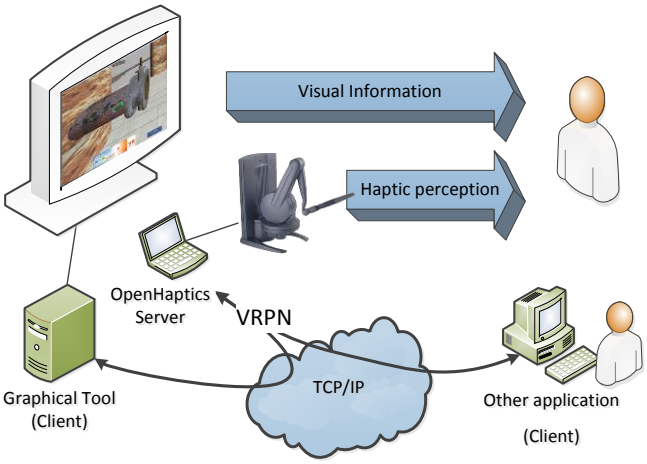

Figure 1: Distributed haptic solution for virtual simulation.

4 OpenHaptics Server: implementation

VRPN is distributed under the Boost Software License 1.0, allowing users to modify this implementation according to their needs. This section details all decisions taken in the implementation of the VRPN OpenHaptics Server.

4.1 Server Architecture

VRPN is a client-server architecture, where the VRPN server runs on the machine associated with the interaction device, and thus it is the responsible of streaming data coming out of the device. The VRPN client consumes the device-emitted data. The architecture allows client and server to run on the same machine or separately on different machines that communicate over a network.

The heart of the VRPN library is the *vrpn_Connection* class. This class is the responsible of sending and receiving messages (using callback handlers) between server and client. The *vrpn_Generic_Server_Object* is a server class that process instructions on which devices to open from a configuration file, allowing developers to customize and expand the VRPN by declaring new VRPN device server classes, such as the one contributed in this work: *vrpn_OpenHaptics* class.

A simplified class diagram of the haptic Server classes is shown in Figure 2. The new device server class responsible for the haptic render loop is called *vrpn_OpenHaptics*, and supersedes the original GHOST-based *vrpn_Phantom* class. The new class inherits from three of the canonical classes: *vrpn_Tracker*, *vrpn_Button* and *vrpn_ForceDevice*, which in turn derive from *vrpn_BaseClass*. The purpose of each class is outlined below.

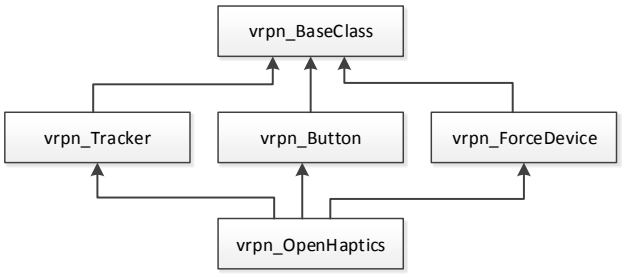

Figure 2: VRPN OpenHaptics Server simplified diagram

- *vrpn_BaseClass* is the parent of all canonic classes. It establishes the connection by way of an instance of the *vrpn_Connection* class. This class handles the registration of message types and contains all methods to pack and unpack the messages that are sent across the connection.
- *vrpn_Tracker* class is the responsible of managing the messages that indicate position, orientation, velocity and acceleration of the Phantom stylus. The implemented haptic server uses the *vrpn_Tracker* standard class, without modifications.
- *vrpn_Button* class provides methods to relay messages concerning the device button state. The *vrpn_button* class requires no modifications from the standard one either.
- *vrpn_ForceDevice* class handles haptic parameter specifications and reports applied force between the haptic device (server side) and the virtual environment (client side). This class has been modified to offer the new interface demanded by the VRPN OpenHaptics server. The extension will be detailed in Section 4.2.
- *vrpn_OpenHaptics* class has direct communication with the device and uses methods supported by the OpenHaptics v3 API. This class performs the haptic rendering, defines structures to hold device data (position, applied force, etc.) and provides these data to the remote client application when any change happens on the server side. The behavior and the interface provided by this new class is detailed in Section 4.2.

4.2 Contributed server classes: (new) vrpn_ForceDevice and vrnp_OpenHaptics.

The implementation presented in this work contributes to the VRPN via a new *vrpn_OpenHaptics* class and a re-implementation of the canonical *vrpn_ForceDevice* class.
The *vrpn_ForceDevice* class, as every canonical class, derives into a remote client interface class and a device server interface class, as shown in Figure 3.
The *vrpn_ForceDevice_Remote* class has two main roles. Firstly, it defines the interface used by the client application to inform the server about the graphic scene. Secondly, it provides a set of callback functions to receive haptic information messages from the server, such as force applied by the user, or device contact-point coordinates. The *Vrpn_ForceDevice_Server* class in turn handles a set of callbacks to receive haptic parameter messages from the client and forwards them to the *vrpn_OpenHaptics* class.

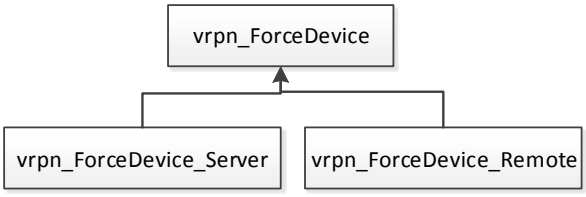

Figure 3: ForceDevice classes

The *vrpn_OpenHaptics* is the device-specific class, implemented from scratch. The implementation is based on the OpenHaptics v3.0 HLAPI layer. First, it defines methods to initialize, close and reset the haptic device. Moreover, the class implements a set of virtual methods declared in the *vrpn_Tracker_Server*, *vrpn_Button_Server* and *vrpn_ForceDevice_Server* classes to receive data to set the haptic scene. Data is stored in specific structures or data types, many of them are OpenHaptics types, and used to carry out the haptic rendering. The class defines a timer to guarantee a 1kHz render frequency. Finally, *vrpn_OpenHaptics* collects all events from the device, such as position, button state and force feedback, and sends them to the client.

Both classes, *vrpn_OpenHaptics* and *vrpn_ForceDevice*, have been written following the VRPN coding rules[1]. In terms of functionality, these classes extend the interface provided by Cuevas-Rodriguez et al. [2013], including methods to (1) implement device vibration, motor vibration and contact vibration; (2) set if a virtual object is touchable or not; and (3) set the workspace bounding box or projection matrix. In terms of performance, the Geomagic haptic servers greatly improve upon the performance of [Cuevas-Rodriguez et al. 2013] by sending all parameters (proxy position, proxy orientation, applied force, depth of penetration, surface contact point, angle at contact point and id of touched object) only when they change.

4.3 Client-Server communication

From the client program side, the haptic device is divided into three different devices as Tracker, Button and ForceDevice, as previously explained. This separation of interfaces allows different types of client applications to access only the information they require. A client class can declare an instance of each remote canonical class (Figure 4) to send and receive each type of data to and from the server.

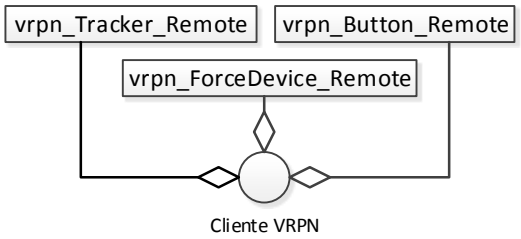

Figure 4: Client-server classes involved in the communication

A typical scenario involves a graphic client that requires total or partial mapping of a graphic scene into a haptic scene. In this context we describe all the interaction required by the client to (1) initialize the haptic scene (from client to server) and (2) interact with the haptic scene (from server to client).

4.3.1 Haptic Scene initialization

Setting up the haptic scene requires the client to follow a three-step process: (1) establish the workspace, (2) define the haptic objects and (3) define the force and vibration effects. The implementation carried out allows flexibility in the order in which these three steps are performed. What follows is a typical initialization scenario.
Once the necessary connections for the operation of the interface have been established, the first step will be to initialize the haptic scene. This is done via the *SetWorkspaceBoundingBox* method or the *SetWorkSpaceProjectionMatrix* method.
To define haptic objects and their properties, the client uses the method called *setObjectNumber* to communicate the number of objects to the server. For each of the objects to define, the client will subsequently send vertex positions, transformation matrix and haptic properties. For this the client should use the *setVertex*, *setTransformMatrix* and *setHapticProperty* methods, respectively. The order in which these parameters are sent is not strict. Figure 5 shows a typical scenario which follows this sequence. All this

[1]VRPN web site Coding rules section (accessed July 26th, 2015): http://www.cs.unc.edu/Research/vrpn/codingRules.html

geometric and haptic information is associated with each of the objects by a unique identifier that identifies each object unambiguously which the client must provide.
The *setTouchableFace* method allows specifying, for all objects defined in the scene, which side will be touchable (front, back or both). This is a global haptic property that must be defined only once and affects all defined objects equally.
Finally, the last step consists in defining the force and vibration effects, if any. These have to be defined via the *setForceEffect* method, which must be called once for each of the effects to be set. All the methods involved can be seen in the Table 1.

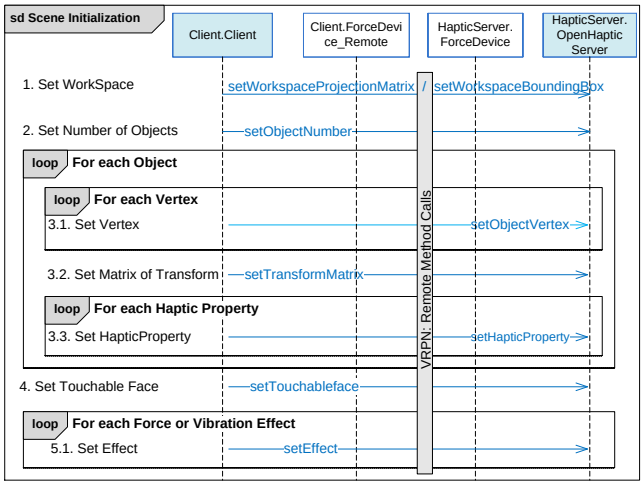

Figure 5: Sequence diagram of a Haptic Scene Configuration

4.3.2 Haptic Scene interaction

Once the haptic scene has been set, the interaction between client and server occurs concurrently through the three different interfaces: Button, Tracker and Force. The Button server reports changes to the state of the device buttons; the Tracker server reports changes to the position and orientation of the device; and the Force server reports all haptic events and receives information from the graphic scene (client). In the following, we describe the interaction between server and client in three through these three interfaces.

Tracker: The tracker interaction is established through the *vrpn_Tracker_Remote* interface, as is usually done in VRPN.
Using this interface the server sends to clients the position and orientation of the haptic device. A common use for this interface is managing the position and orientation of the graphical representation of the haptic device in the graphic scene.

Tracking information is not always sent; only when the device is moving the server calls the *sendProxyPosOrient* method to avoid saturating the network with irrelevant information. Figure 6 shown this scenario. The server considers that the device has moved when configurable thresholds in linear or angular distance are reached. Clients subscribe to this information via a callback mechanism.

Table 1: Methods added to vrpn_ForceDevice_Remote standard distribution class to configure a Haptic Scene.

Methods	Description
setObjectNumber	Sets the number of objects to render
setObjectVertex	Sets the vertex of an object
setTransformMatrix	Sets the transformation matrix for each object, which provides orientation, position, and scale of the object.
setHapticProperty	Sets the haptic properties of an object: stiffness, damping, static and dynamic friction, pop through, and mass.
setTouchableFace	Sets the face of the object that will be haptically rendering: front, back, or both. This feature is the same for all objects.
setWorkspaceProjectionMatrix	Sets the workspace based in the model and projection matrix. The workspace is the space where the device is going to interact with the VE.
setWorkspaceBoundingBox	Sets the workspace based in the model matrix and a bounding box.
setEffect	Sets the force effects to render. HLAPI provides four effects: constant force, spring, viscosity, and friction. Besides, we have added two vibration effects, called motorVibration and contactVibration. For each one is necessary provide a few of this parameters: gain, magnitude, frequency, duration, position and direction. See Table 4.

Button: The button interaction is established through the *vrpn_Button_Remote* interface. By using this interface, the server sends to clients status information of the haptic device buttons. Similarly to the Tracker server, only changes on button state are sent to clients.

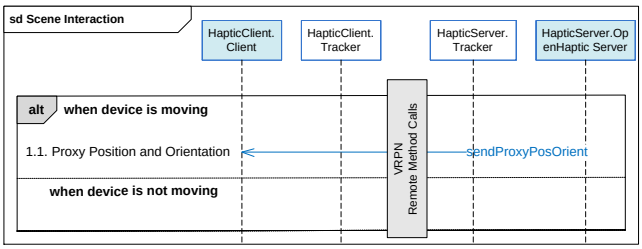

Figure 6: *Sequence diagram of Tracker Interaction*

Force Device: The force interaction is established through the *Vrpn_ForceDevice_Remote* interface. Its methods for haptic scene interaction are organised in two groups: (1) A set of VRPN callbacks, which allow the client to have access to the *send* functions defined in *vrpn_OpenHaptics*; through these, the server is able to send back to the client all the information about the haptic events on the scene (Table 2 details these methods) (2) The set methods (Table 3) through which the haptic client sends to the server the information required to modify the scene.

Table 2: Send Methods, defined in *vrpn_OpenHaptics*

Methods	Description
sendForce	Sends the applied force
sendDOP	Sends the Depth of Penetration
sendSCP	Sends the Surface Contact Point
sendIsTouching	Indicates if it is touching an object
sendTouchedObject	Sends the identified of the object that this is touching
sendAngle	Sends the angle at contact point

Figure 7 shows a typical scenario of interaction between a client and haptic scene divided in the two blocks mentioned before: (1) messages that flow from server to clients through the callback & *send* interface, informing all subscribed clients about events happening in the haptic environment. When a contact occurs the *vrpn_OpenHaptics* server uses these methods to broadcast information. When the contact stops, the server stops sending data, with the only exception of the *sendIsTouching* method, which sends continuous information to indicate clients whether any contact is taking place. And (2) messages emerging from the client (the graphic scene) to the server to modify the workspace; turn on/off haptic rendering of an object; turn on/off a force or vibration effect; and reset the scene at a given time.

Table 3: Methods added to vrpn_ForceDevice_Remote standard distribution class to interacting with the Haptic Scene

Methods	Description
startEffect	Indicates that the force effect should begin.
stopEffect	Indicates that the force effect should finish.
setObjectTouchable	Indicates that the server should start/stop render an object. By default, the server renders all objects.
resetScene	Reset the Haptic Scene. The server deletes workspace, objects and effects.

4.4 Contributed Force Effects

The server developed offers two custom vibration effects, which are not offered directly by OpenHaptics. The effects have been developed in the context of industrial simulations and make use of the low level HD libraries to operate. The effects are called *Motor vibration* and *Contact Vibration*. Both of them are independent vibration effects, which can operate together or independently during the VR simulation. And both effects are parameterized by two parameters, magnitude and frequency, which have to be set by the client.

Once these effects have been defined, they may be activated when desired by the client, through the corresponding interface. Once they are activated, the server sets them in the OpenHaptics rendering thread, using an OpenHaptics callback. In the following we detail each of the vibration effects:

Motor Vibration: This effect is intended to render vibrations such as those produced by an electric motor, allowing simulation devices such as may be drills, polishers, grinder, etc. Once activated, this effect of vibration is felt at all times until deactivation. The vibration is generated by way of a sinusoidal force effect with two components, F_x and F_y on the x-axis and y-axis of the device, respectively:

$$F_x = M\cos(2\pi\varphi t)$$
$$F_y = M\sin(2\pi\varphi t)$$

where M and φ are the magnitude and frequency values set by the client.

Contact Vibration: This vibration effect is intended to simulate the vibrations that would occur by touching a surface with one of the devices simulated by the other effect (e.g. rills, polishers, grinders, etc.) This effect, once activated, is not operating continuously. It only comes into operation when a virtual object is touched with the device. The vibration is generated by way of a sinusoidal force effect in the direction normal to the contact surface. The value of this vibration is: $F_N = M\sin(\varphi t)$.

5 Using VRPN OpenHaptics Server

The development carried out may be of interest for any VR application to which one would want to incorporate haptic interaction. The main steps to incorporate haptics would be to (1) compile the VRPN server for the required platform and (2) to incorporate a VRPN client in the application. As explained in Section 4.3, through this client the application first sends all the geometric and haptic information to configure the haptic scene and then receives all haptic events. Let us look at how one would do this.

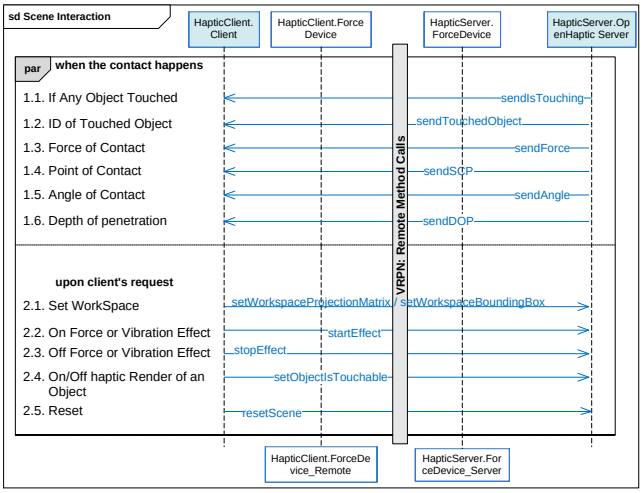

Figure 7: *Sequence diagram of a Scene Interaction*

```
#include <vrpn_ForceDevice.h>

void VRPN_CALLBACK handle_force_change(void *userdata, const vrpn_FORCECB f)
{
        printf("Force:%lf,%lf,%lf\n",f.force[0],f.force[1], f.force[2]);
}
int main(int argc, char **argv)
{
        ForceDevice = new vrpn_ForceDevice_Remote(server);
        // Wait until we get connected to the server.
        while (!ForceDevice->connectionPtr()->connected())
        {
                ForceDevice->mainloop();
        }
        //Sending Scene
        ……
        //Register Remote Functions
        ForceDevice->register_force_change_handler(NULL, handle_force_change);
        ……
        // main loop
        while ( state != quit )
        {
                ForceDevice->mainloop();
         }

}
```

Figure 8: Complete program to subscribe to the force method of the *Send* interface and print the force received

5.1 Client side

Follow the next steps to add the VRPN client inside your application and access the services provided by vrpn_OpenHaptics.

Instantiate the remote classes: Include the following VRPN header files *vrpn_ForceDevice.h*, *vrpn_Tracker.h* and *vrpn_Button.h* to be able to instantiate objects of the remote classes *vrpn_ForceDevice_Remote*, *vrpn_Tracker_Remote* and *vrpn_Button_Remote*. From that moment, the application will be able to make use of the methods of these classes. The contribution presented in this paper is focused on the ForceDevice and ForceDevice_Remote classes. Hence, in the following we detail how to use the methods of these classes; for the other interfaces VRPN refer to the VRPN official documentation[2]. In Figure 8 an example to subscribe to the force method of the ForceDevice classes is shown.

Start the haptic scene: The right time to start the haptic scene is just after the application has built the graphic scene. Once this has happened the application can make use of the services provided by *vrpn_OpenHaptics* to: (1) configure the haptic scene, (2) interact with it.

Send the objects that form the haptic scene: Once the haptic scene has been started, the first thing that must be sent is the number of objects in the graphic scene to be rendered haptically. For each object, its vertices, and its transformation matrix and haptic properties have to be sent too.

Configure the workspace: The application must send the workspace too. This workspace can be defined based on the model matrix and the projection matrix of the graphic scene, if one wishes to render haptically the whole graph scene. Alternatively, it can be defined based on the model matrix and a bounding box, if one wishes to render haptically only a part of the graph scene. The necessary methods to do this scene configuration have been detailed in Table 1 and Section 4.3 of this paper.

The system developed does not need a signal to start the haptic render. On the contrary, the haptic render is working from the beginning. Consequently, once the objects and the workspace have been defined it is possible to touch the objects by the device. The reception of all these parameters by the server involves a high computational load. Thus during configuration the server is not able to guarantee correct timings in the haptic render. It is for this reason that we have distinguished two stages in the communication between haptic server and client (Section 4.3).

The configuration of the force effects should not slow down the server, unless many of them will be defined consecutively. Still, in order to ensure the correct haptic rendering is better to do it now.

Define haptic effects: As a result, next step in the configuration of the haptic scene is to define the effects of force or vibration, if any. The server is able to render six different force effects. Four of (constant, spring, viscous and friction) are directly offered by OpenHaptics, please refer to the OpenHaptics documentation for more info[3] . The other two vibration effects (contact-vibration and motor-vibration) are added by this sever, and they are not define in the OpenHaptics libraries. To define an effect, the graphic application should send a set of parameters (see Table 4). For the vibration effects, these are a magnitude parameter to indicate the

[2] VRPN main page (accessed July 26th, 2015): http://www.cs.unc.edu/Research/vrpn/index.html

[3] OpenHaptics Programmer's guide (accessed July 26th, 2015):http://www.geomagic.com/files/4013/4851/4367/OpenHaptics_ProgGuide.pdf

Table 4: Parameters involved in rendering each of the effects.

Effect Type	Gain	Magnitude	Duration	Frequency	Position	Direction
constant	-	X	X	-	-	X
spring	X	X	X	-	X	-
viscous	X	X	X	-	-	-
friction	X	X	X	-	-	-
contactVibration	-	X	-	X	-	-
motorVibration	-	X	-	X	-	-

maximum force that the effect can generate and the frequency of the vibration.

Proceed with haptic rendering: Once the haptic scene has been completely established, the server can guarantee the correct haptic render of the scene. The user can now interact with the haptic and graphic scene. For this the graphic application should interact with the haptic server through the *set* methods and *send* methods provided through the *vrpn_ForceDevice_Remote* interface. Through the *set* methods the client can send orders to the server, to indicate different actions to be executed such as start or stop of a force effect. Through the *send* methods the server informs the client of the haptic events that occur. These methods are called by the server by the use of VRPN callbacks. Table 2 shows the methods implemented in this server. By them the server will inform when a contact happens, what force is being fed back by the device and so on. The client has to subscribe to all or only any of them, depending on its needs.

5.2 Server side

On the server side not many actions are required. In order to deploy the server, one must download the VRPN project and follow the instructions to compile the *vrpn_server*. It is mandatory to activate the *cmake* options to add the VRPN OpenHaptics packages and *vrpn_OpenHaptics* to the compilation. Once the server is compiled, one should modify the vrpn.cfg file making sure that the line to start the OpenHaptics server is present.

6 Performance

In a VR system, latency is a critical parameter; high latency can cause offset, glitches and other bad user experiences. In this section we present tests performed to check that at least the VRPN performance was not altered by the introduction of this new server. In order to assess the proposed server performance, it has been measured the time it takes a haptic event to reach the VRPN client since it is generated in the VRPN server (t_{VRPN}). While in a stand-alone, non-distributed application, where haptic and graphical loops are directly connected, this time is negligible, in a distributed application, in which haptic (server) and graphic (client) render run in different machines, this time could become in the main source of delays. Nevertheless, to provide a fair comparison, network latency was excluded of the measurement. For this reason, time was measured with both server and client running on the same machine, but using the whole VRPN protocol to transfer information, i.e both client are server are on *localhost*. As a result, only the latency due to the VRPN overhead has been considered in t_{VRPN}.

The test was carried out on an Intel Core 2 Quad Q8400 at 2.66 GHz with Windows 7. We measured the time elapsed between the generation of the haptic event by OpenHaptics, and its arrival to the client. In particular, the test was done using the *sendTouchedObject* interface. During the test a total of 49200 samples were taken. 99.91% of them took less than 0.5ms to reach the client from the haptic server (see Table 5).

Table 5: Number of samples with very long t_{VRPN}.

Elapsed time	Samples	%
<300 µs	49041	99.68%
300-500 µs	115	0.23%
501-1000 µs	33	0.07%
>1000 µs	11	0.02%
Total	49200	

Figure 9 depicts the histogram of elapsed time from server to client (t_{VRPN}) in terms of occurrences, in order to illustrate the server performance. The average elapsed time is a bit over 100µs (SD < 50µs). During the test only 11 samples took more than 1ms to reach the client. Although this is still a very small percentage, it could cause glitches during the VR simulation. A deeper analysis is necessary to understand whether these are related to the experimental conditions.

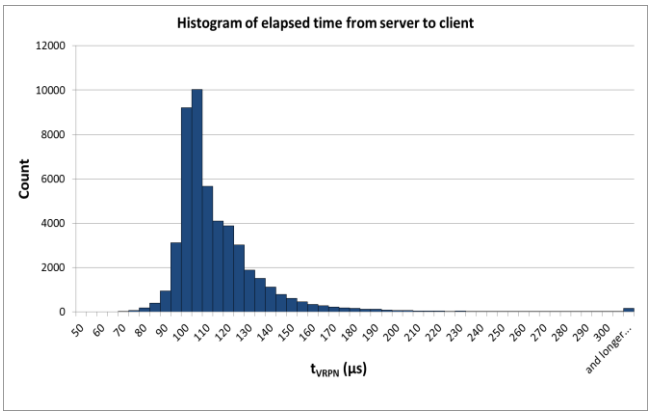

Figure 9: *Histogram of t_{VRPN} (Elapsed time from a haptic event to reach the graphical client).*

7 Conclusions

The Virtual Reality Peripheral Network (VRPN) library was developed almost fifteen years ago, but we believe it is still a good idea, and so do most of the providers of tracking technology. When the VRPN is used to connect VR applications and haptic devices, the graphic and the haptic render loops can be developed separately which reduces developing effort and distributes computational load.

We have presented a contribution to the (VRPN) aimed at updating its support of the Geomagic (formerly Phantom) family of haptic devices. The contribution follows VRPN coding rules, creating a new *vrpn_OpenHaptics* server and extending the *vrpn_ForceDevice* canonical interface.

The implementation allows both full and partial mapping of a graphic scene in a haptic environment. Arbitrary geometries can be haptically rendered by full use of the OpenHaptics HLAPI libraries. The now obsolete GHOST libraries are no longer required by our implementation, which offers a much richer interface to clients. This allows greater control of the haptic scene, and facilitates development of more advanced VR applications. Still, the provided interface is simpler to the full OpenHaptics API, reducing developing effort for most applications. The sever adds two new vibration effects which are not present in OpenHaptics.

The current version of the new vrpn_OpenHaptics sever only allows configuration of static scenes. Our future plans include the simulation of rigid body dynamics in the client side, which requires sending regular updates of rigid transformations of individual objects from the graphical client to the haptics server. Simulating deformable objects is also feasible, but a naïve approach would imply sending frequent updates of multiple points position (depending on the discretization used to compute deformation) and optimizations would be needed to avoid high band-width consumption and inconsistencies due to the time needed to send the deformation state of each single object. To improve scalability of the server for these and other future improvements, we are working in a middleware based on VRPN for multi-rate synchronization of the server and the different clients (graphical and other), avoiding the need to send too frequent (1kHz) updates to clients working at much lower rates.

References

CUEVAS-RODRIGUEZ, M., POYADE, M., REYES-LECUONA, A., MOLINA-TANCO, L. 2013. A VRPN server for haptic devices using OpenHaptics 3.0. In New Trends in Interaction, Virtual Reality and Modeling (pp. 73-82). Springer London.

FREEMAN, D., PUGH, K., VORONTSOVA, N., ANTLEY, A., SLATER, M. 2010. Testing the continuum of delusional beliefs: an experimental study using virtual reality. Journal of abnormal psychology, 119(1), 83.

ITKOWITZ, B., HANDLEY, J., ZHU, W. 2005, March. The OpenHaptics™ toolkit: a library for adding 3D Touch™ navigation and haptics to graphics applications. In Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint (pp. 590-591). IEEE.

JACOBSON, J., LE RENARD, M., LUGRIN, J. L., CAVAZZA, M. 2005, JUNE. The CaveUT system: immersive entertainment based on a game engine. In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (pp. 184-187). ACM.

MARSHBURN, D., WEIGLE, C., WILDE, B. G., TAYLOR, R. M., DESAI, K., FISHER, J. K., SUPERFINE, R. 2005, October. The software interface to the 3D-force microscope. In Visualization, 2005. VIS 05. IEEE (pp. 455-462). IEEE.

OTADUY, M., GARRE, C., LIN, M. C. 2013. Representations and algorithms for force-feedback display. Proceedings of the IEEE, 101(9), 2068-2080.

RENARD, Y., LOTTE, F., GIBERT, G., CONGEDO, M., MABY, E., DELANNOY, V., BERTRAND O., LÉCUYER, A. 2010. OpenViBE: an open-source software platform to design, test, and use brain-computer interfaces in real and virtual environments. Presence: teleoperators and virtual environments, 19(1), 35-53.

SUMA, E., LANGE, B., RIZZO, A. S., KRUM, D. M., BOLAS, M. 2011, March. Faast: The flexible action and articulated skeleton toolkit. In Virtual Reality Conference (VR), 2011 IEEE (pp. 247-248). IEEE.

TAYLOR II, R. M., HUDSON, T. C., SEEGER, A., WEBER, H., JULIANO, J., HELSER, A. T. 2001, November. VRPN: a device-independent, network-transparent VR peripheral system. In Proceedings of the ACM symposium on Virtual reality software and technology (pp. 55-61). ACM.