EG UK Theory and Practice of Computer Graphics (2010)
John Collomosse, Ian Grimstead (Editors)

Integrating Haptic Interaction Into An Existing Virtual Environment Toolkit

Peter Scopes and Shamus P. Smith†

School of Engineering and Computing Sciences
Durham University, Durham, United Kingdom

Abstract

There is increasing demand for haptic, or touch-based, interaction in virtual environments. Although many haptic devices come with APIs to enable the development of haptic-based applications, many do not provide the same level of graphical support available in virtual environment or game technology toolkits.
This paper will discuss the integration of haptic interaction into an existing virtual environment toolkit. By creating a flexible middleware component, haptic interaction and force feedback for a haptic device can augment sensory experiences in existing virtual environments. A user study was conducted to evaluate the integration of haptics and realistic physics in an example virtual environment.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Methodologies]: Computer Graphics[three-dimensional graphics and realism] D.2.13 [Software]: Software Engineering[reusable software]

1. Introduction

Virtual environments are commonly visually oriented. A contributing factor has been the technological constraints on providing a richer sense-based environment [Kal93]. Over the last decade visual technologies have matured and it is now common to find virtual environments using a range of visual-oriented technology, for example desktop monitors, head-mounted displays and surround-screen displays [BKLP05].

Interaction in such virtual environments, and the potential for complex and realistic interactions, has been driven by the dominance of these visual technologies. This has limited the veracity of the resulting interaction [Smi07]. Stanney et al. [SMK98] note that "multimodal interaction may be a primary factor that leads to enhanced human performance for certain tasks presented in virtual worlds." Unfortunately to be able to touch or *feel* virtual objects has been constrained to bulky and expensive equipment. However, recent advances in technology has meant that haptic devices are now being used in all manner of applications [WB06] including medical device simulation, computer aided design, visualisation and the graphic arts [BS02]. Virtual environments using sight and touch are quite feasible, but there is a dearth of resources for building such multimodal environments.

Although there are a number of virtual reality (VR) toolkits available, most prioritise graphical rendering, usability issues and the support of 3D graphical models. It is less common to find general VR toolkits that also support sensory feedback as provided by haptic cues. A contributing factor has been the immaturity and availability of the associated output technology. However, reduced cost hardware is now available to support novel and complex interactions as seen in the use of 3D spatial interaction using the Nintendo Wii [LaV08] and force feedback interaction with affordable haptic devices such as the PHANTOM Omni (http://www.sensable.com) and the Novint Falcon (http://home.novint.com). It is common for support for such technology to be by the hardware developers, although there is increasing interest in device independent APIs for use in existing graphical applications, such as VR toolkits and gaming engines [DP07].

This paper describes the integration of haptic, or touch-based, interaction into an existing virtual environment engine. A framework was developed to allow developers to

† Correspondence author: shamus.smith@durham.ac.uk.

DOI: 10.2312/LocalChapterEvents/TPCG/TPCG10/241-248

transparently integrate haptic cues into virtual environment applications. A number of example virtual environment and haptic APIs were reviewed and a middleware library was constructed. A user study was conducted to evaluate the user perceived realism and accuracy of the haptic integration.

2. Related Work

Much of the related work has focused on the development of environments to integrate visual and haptic representations. These systems, which are typically sets of C++ libraries, aim to move away from the traditional use of low-level haptic device drivers or APIs supplied with specific devices, for example those provided by SensAble Technologies, developers of the PHANTOM range of haptic devices (http://www.sensable.com), and Reachin, developers of the *Reachin Display* supporting co-located visual/haptic experiences (http://www.reachin.se). This section will overview a number of relevant examples of visual/haptic development environments and some recently developed haptic library technologies.

CHAI 3D (http://www.chai3d.org) is a freely available set of C++ libraries for graphics and haptic rendering which has built-in support for multiple haptic devices. CHAI 3D is designed to tightly integrate haptics and visual representations of objects and to remove the complexities of individual haptic devices. The graphics rendering is built over OpenGL and allows mono and stereo graphics. This system is an example of an integrated development environment which can be used to construct multimodal virtual worlds. By integrating the visual and haptic representations of objects into a single set of libraries allows a developer to build applications at a higher abstraction level than when working with individual haptic devices.

The **H3D API** (http://www.h3dapi.org) is an open source haptics software development platform that uses the open standards OpenGL and X3D (http://www.web3d.org/x3d) with haptics in one unified scene graph to handle both graphics and haptics. It aims to be both cross platform and device independent while supporting a reduction of data redundancy as it can render graphics and haptics from the one scene graph.

The **OpenSceneGraph** (http://www.openscenegraph.org) (OSG) project is a scene graph graphics toolkit designed for the development of high-performance graphics applications. OSG is open source and provides an object oriented framework on top of OpenGL. This supports rapid application development as the developer is not restricted to low-level graphics calls. The OSG project aims to make scene graph technology available to both commercial and non-commercial users. Key strengths include performance, scalability and portability.

De Pascale and Prattichizzo [DP07] describe the **Haptik Library** (http://www.haptiklibrary.org). This system was developed to provide easy but powerful low-level access to haptic devices from different vendors. It has a component based architecture and supports plug-ins to support new devices and drivers to be added without recompilation of the library or existing client applications. De Pascale and Prattichizzo demonstrate a number of alternative plugins for Haptik and note its use in a robotic system. The library was evaluated by considering callback timings to a PHANTOM haptic device and they concluded that the Haptik library runs at least as fast as applications using low-level haptic libraries, such as the CHAI API. Their work does not detail any user performance evaluation of the Haptik Library.

Pava and MacLean [PM04] consider a middleware architecture for distributed implementations across multiple computers on a network to support haptic rendering and visualisation processes. Their system provides an extendable, device-independent and network-transparent interface to I/O devices. They evaluated their system with timing experiments which demonstrated the feasibility of using a distributed object computing architecture for multimodal applications.

3. Integrating Haptic Interaction

There are many different virtual environment engines and toolkits available. In the context of this work, we reviewed a number of different systems which were either open source or free-ware. A number of dimensions to support initial graphical engine/toolkit selection were determined including (i) programming language used, (ii) the native system language, and (iii) any evidence of documentation, tutorials and an online development community. Section 3.1 overviews a number of candidate graphical systems and Section 3.2 overviews candidate haptic APIs.

3.1. Virtual Environment Engines

The **Blender Game Engine** (Blender GE) was developed from Blender (http://www.blender.org), a 3D content creation suite. Blender is an open source modelling and animation software and is free under the GNU General Public License (GPL). The game engine is embedded into the Blender program and as such isn't the main focus of the program. Blender is mainly a modelling and animation program, though the functions and capabilities of the game engine are growing with each release of the system. The documentation for the GE is a work-in-progress. Unlike most other virtual environment engines, the Blender GE is much more focused on 3D models and graphical interfaces as opposed to the programming, though scripting in the form of Python is possible. To support haptic interaction it is likely that it would require re-writing parts of the engine kernel. Although this would allow for a more full integration of the haptics into any developed virtual environments it would require expert knowledge of the kernel, with associated time implications.

OGRE (http://www.ogre3d.org) (Object-Orientated Graphics Rendering Engine) was built from, and is being maintained by, a small core development team support with a large group of independent contributors. OGRE is open source and is free under the GNU LGPL. As the name suggests it is not a game engine, a modelling program nor an all-in-one solution. The ideology behind OGRE is that it is to be used as a component in a custom virtual environment engine. The aim is not to force developers to use a specific modelling program or game logic but to allow developers to choose the best components that fit their current project. There is a large community of users to help support other users. OGRE is a very customisable tool, written in C++ and can be compiled onto the different platforms including Windows, Linux and Mac OSX. It has a component-based design which would allow plug-ins or libraries to be written to integrate haptics. Although it is open source, and thus it would be possible to re-write the kernel, the complexity and the component based nature of OGRE could make this problematic.

The **jMonkey Engine** (http://www.jmonkeyengine.com) (jME) is a scene graph based API created to be a high performance graphics engine for the Java language. It uses an abstraction layer to allow any rendering system to be plugged in. jME is open source under the BSD (Berkeley Software Distribution). jME is focused on allowing the use of all the features that graphics cards can give and making it easier for developers to use them. As it is written in and for Java, it is thus naturally multi-platform. Although the modular approach to design and the object oriented (OO) structure of Java would help in the development, many haptic devices and haptic APIs are written in C++. This could add complications in developing middleware or direct kernel modification for haptic device support.

Java3D (https://java3d.dev.java.net) is a scene graph based API which runs on top of either OpenGL or Direct3D. It provides a high-level interface for creating and manipulating 3D geometries and building structures used in rendering geometry. Java3D is an open project, but not open source, released under the BSD licence. Java3D is written in Java and hence has the same multi-platform functionality as the Java platform. As it is common for haptic devices to be written in C++, this would make integrating the haptics with Java3D challenging. Although Java boasts the JNI (Java Native Interface) which allows other languages, such as C, C++ and assembler, to call Java code that is running in the Java virtual machine, Java3D is not open source and thus it would not be possible to modify or make changes to the underlying kernel.

MAVERIK (http://aig.cs.man.ac.uk/maverik) [HCK*01] is a VR toolkit developed by the Advanced Interfaces Group (AIG) at The University of Manchester (UK). MAVERIK is designed to primarily deal with graphical and spatial management allowing it to support, among other things, high-performance rendering. Due to its design MAVERIK provides a stable environment for rapid production of complex virtual environments as well as many functions that are useful for developing applications with 3D graphics and/or using 3D peripherals. MAVERIK is made up of two components, the micro-kernel which implements a set of core services and framework, and the supporting modules which contain methods for various functionalities. MAVERIK is open source and free under the GNU GPL licence. MAVERIK uses a novel approach to data representation. Instead of converting the data into a native format, like other VR systems, it avoids this by using its own internal data structures which leads to two important benefits. It can easily take advantage of optimisations that are highly application specific and it can readily dynamically adapt to a wide range of application demands. It is written entirely in C++ which will allow a library to be written to directly incorporate haptics or alternatively the kernel could be altered and recompiled.

The **Irrlicht engine** (http://irrlicht.sourceforge.net) is a high performance 3D engine with a high level API for creating complete 2/3D applications like games and scientific visualisations. It is free and open source under its own license based on the zlib/libpng license. Irrlicht is purely a graphics engine and as such has no sound built into it but offers a free plug-in called irrKlang (http://www.ambiera.com/irrklang) and offers other tools to support development. It is platform independent and rendering can be done using Direct3D, OpenGL and some of its own renderers. It also supports a large range of different input file formats and bindings which makes the engine available to all .NET languages. Irrlicht comes with a variety of common special effects, such as realistic water surfaces, and dynamic shadows. It is written entirely in C++ and boasts full object oriented design. Since it is open source, the kernel of the engine could be re-written to allow for haptic integration as could an extension library be written or plugged in.

Coin3D (http://www.coin3d.org) is a high-level 3D graphics toolkit designed to be fully backward compatible with the Open Inventor API since Open Inventor 2.1 is a de facto standard for 3D visualisation and visual simulation. Coin3D is portable over many platforms and is under duel release licenses GNU LGPL and PEL (Professional Edition License). It is built on top of OpenGL and uses scene graph data structures to render the 3D graphics. It also implements user interface bindings to open window management which it does through a number of libraries and comes with several file I/O libraries. Coin3D saves CPU resources by only redrawing when the scene data is changed making it better suited for user interface-based applications. Coin3D is able to work with OpenGL allowing a gradual change over from an application purely in OpenGL to Coin3D. Since Coin3D is available under the GNU LGPL for free-software this would allow the source code to be altered to integrate

haptics. An external library/toolkit could also be written to integrate haptics into this graphics engine.

3.2. Haptic APIs

The **OpenHaptics toolkit**, from SensAble (http://www.sensable.com), aims to give software developers the ability to add haptics into a wide range of applications, from games to simulations and visualisations. The toolkit was patterned after OpenGL to support learnability for graphics programmers. It provides haptics commands so that developers can use existing code for specifying geometry to give properties such as fiction and stiffness. OpenHaptics is designed to be extendable so that extra functionality can be implemented and third-party libraries can be integrated. The toolkit only supports SensAble haptic devices, e.g. the PHANTOM range of haptic devices, but is cross-platform. The toolkit provides two libraries, the Haptic Device API (DHAPI) which a low-level API and the Haptic Library API (HLAPI) which is a higher level API. These two libraries aim to allow users of the toolkit to gain precise control or a higher level abstraction as needed.

HAPI (http://www.sensegraphics.com) is a haptics rendering engine which is device independent with built-in support for multiple haptic devices. It is open source and cross platform and released free under the GPL license. HAPI is written entirely in C++ and was created in the development of H3D, a scene-graph API (see Section 2). It is designed to be modular and all the features can be extended or modified by users. It allows force generation from 3D graphics and is tested with both DirectX and OpenGL. HAPI gives the user collision handling, haptics rendering algorithms, surface interaction, force effects and thread handling. HAPI is both a good example of a haptics API and a good candidate to be used in the integration of haptics into an existing virtual environment. It allows developers to work at a higher level of abstraction than other haptics APIs, e.g. the OpenHaptics toolkit. As it is open source, modifications would be possible if the need arose to implement extra functionality.

DHD-API (http://www.forcedimension.com) is a haptics API from Force Dimension which, by hiding the complexities of haptic programming, enables users to add haptic capabilities to their applications quickly and easily. It allows users, when required, to use a range of low-level calls to provide advanced control over haptic devices. It is available on all major operating systems. It has a modular design providing a single programming interface with consistent syntax and also has built in third-party support for other haptic visualisation packages. The DHD-API is a commercial product from Force Dimension which is written in C/C++. It is not a candidate for the haptic API that will be used as it is not a free API and it is very much aimed at Force Dimension products.

3.3. Virtual Environment Engine and Haptic API Selection

Even within the small subset of virtual environment engines that have been considered in Section 3.1 there are a number of potential candidates, such as MAVERIK and Irrlicht, to choose from to integrate into a haptics API. A summary of the key features of each of the reviewed virtual environment engines can be seen in Table 1. Although many of the virtual environment engines are suitable candidates, OGRE was chosen because it is very powerful, customisable, designed for plug-ins and has a large active development community.

There are fewer freely available haptic APIs. This is unsurprising considering that haptics is a relatively new field and there isn't such a large commercial backing as there is behind graphics nor are haptic devices commonly in use. The OpenHaptics toolkit only works with SensAble products which would limit the usage of any middleware using this. This restriction isn't necessary as there are other APIs which offer the same service but also allow for other product lines. DHD-API is able to support more haptic devices than just those from the manufacturers' product line which would therefore be desirable but as it is not free, this leaves HAPI as the choice for the haptic API to use. It has number of advantages including that it is free, supports third-party haptic devices and it provides a high level interface. Therefore we have developed a set of middleware libraries to sit between OGRE and HAPI called *OgreHAPI*.

4. OgreHAPI

There are generally three alternatives when attaching haptic devices to a virtual environment engine/toolkit. Either (i) write a new device driver, (ii) use a device specific API or (iii) use a device independent API. Also if the virtual environment is open source then any of these solutions could be written directly into the kernel code, requiring a recompilation of the kernel, or alternatively via a higher level abstraction layer, e.g. a toolkit API (see Figure 1).

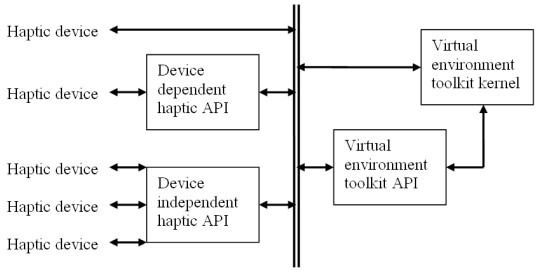

Figure 1: *Integrating haptic devices into virtual environments.*

We chose to implement OgreHAPI as a middleware component between a haptic API and a virtual environment

	Languages	Community	Documentation	Tutorials	Native code
Blender GE	Python	Large, active	Work-in progress	Yes	C++
OGRE	C++	Large, active	Yes	Yes	C++
jMonkey	Java (C,C++)*	Large, active	Java doc	Yes	Java
Java3D	Java (C,C++)*	Large, active	Java doc	Yes	Java
MAVERIK	C,C++		Yes	Demos	C++
Irrlicht	C++	Large, active	Yes	Yes	C++
Coin3D	C++	Mailing lists	Yes	Yes	C++

Table 1: *Summary of virtual environment engine features (* available through JNI).*

toolkit API as integrating directly into the kernel of a virtual environment engine would require a high level of expertise of the engine and could result in undesirable, and hidden, consequences due to dependability issues between components in the kernel. Also as both OGRE and HAPI are third-party applications, changing either kernels would irrevocably customise the final system. This is against the ethos of both OGRE and HAPI. Finally, working at a kernel level would be very time consuming as building the kernel of OGRE takes 20+ minutes to compile.

In the simplest form OgreHAPI works by attaching *Objects*, which describe the haptic shape, position, orientation, etc. to OGRE scene nodes. These objects are then put into the haptic *Space* so that they will be rendered. A *FrameListener* ensures that before each frame is rendered the positions and orientations of all the *Objects* in the haptic *Space* are updated to match the scene node that they are attached to. Furthermore the *FrameListener* handles input device input (such as keyboards) and updates the position and orientation of any environment avatar representing the haptic device in the OGRE world.

A call to update positions and orientations will also check to see if the haptic device is within the bound object of the *Object*. If so, it will render it relative to the position and orientation of the camera in which the OGRE world is being observed. This way the user can explore the world both visually and haptically.

The OgreHAPI library is made up of five main classes.

- OgreHAPIPrimitives: Pre-built primitive OgreHAPI objects including *Sphere* (a primitive haptic shape), *MeshObject* (a haptic shape based on a specified mesh) and *MeshObjectAnimated* (a haptic shape based on a specified mesh with animation).
- OgreHAPIObject: This is a haptic object to be rendered. This class keeps track of position, orientation, shape, etc. of the haptic object.
- OgreHAPIFramelistener: Keeps the haptic rendering update-to-date with the specified *SceneManager* projecting the rendering so that it is relative to the camera position and orientation. It provides a means of controlling camera movements through unbuffered input from the keyboard.
- OgreHAPISpace: Tracks all the haptic objects that should be rendered, for example with functions to add haptic objects to a space or to query the details of all the haptic objects currently in a space.
- OgreHAPIUtil: The link between OGRE and HAPI. Converts units between OGRE [Vector3, Quaternion, Radian, Matrix4], HAPI [Vec3, Matrix4] and H3DUtil [Quaternion, Vec3f, H3DFloat angle, Rotation]. Also translates specified vector positions from relative positions to global positions and vice versa.

There are three main steps for building an OgreHAPI application. The following sections will use an example application, called *What's In The Box* (also see Section 5.1). These sections only provide an overview of this process (OgreHAPI documentation, classes and two tutorials are available online at http://www.dur.ac.uk/shamus.smith/ogrehapi/).

4.1. Step One: Set Up a Header File

The first header item is a sub-class of the *OgreHAPI::FrameListener* class with a single extra variable *mApp* which is a pointer to the main application, in this case *WhatsInTheBoxApplication*. The constructor is then overridden to initialise *mApp* when creating a new instance of the *WhatsInTheBoxFrameListener*:

```
class WhatsInTheBoxFrameListener
  : public OgreHAPI::FrameListener
  {private: WhatsInTheBox *mApp;};
```

The next constructor is exactly the same as the standard *OgreHAPI::FrameListener* constructor except for the addition of the pointer to the *WhatsInTheBoxApplication*.

```
public:
WhatsInTheBoxFrameListener(Ogre::RenderWindow
  *mWindow, Ogre::SceneManager *mSceneMgr,
  Ogre::Camera *mCamera,
  HAPI::HAPIHapticsDevice *mHapticsDevice,
  OgreHAPI::Space *mSpace,
  WhatsInTheBoxApplication *app);
```

The next constructor calls the *OgreHAPI::FrameListener* constructor as well as collecting the pointer for the *WhatsInTheBoxApplication* object. It also calls a function

called *setCameraOffset* which sets, in OGRE units, the position of the center of the haptic device relative to where the camera is. The *setScale* function sets the scale of the number of meters that one OGRE unit should be.

OgreHAPI supports the movement of the camera around the OGRE world, thus allowing users to change what they are able to haptically interact with. OgreHAPI also provides a function *setFixedCamera(bool fixed)* which can fix the camera so that, unless code elsewhere is written to move it, the camera won't be able to move.

```
WhatsInTheBoxFrameListener::
  WhatsInTheBoxFrameListener(Ogre::
  RenderWindow *mWindow,
  Ogre::SceneManager *mSceneMgr,
  Ogre::Camera *mCamera,
  HAPI::HAPIHapticsDevice *mHapticsDevice,
  OgreHAPI::Space *mSpace,
  WhatsInTheBoxApplication *app) :
  OgreHAPI::FrameListener(mWindow, mSceneMgr,
  mCamera,mHapticsDevice, mSpace),mApp(app)
  {setCameraOffset(Ogre::Vector3(0,0,30));
    setScale(0.01); }
```

4.2. Step Two: Set Up the Main Application

The main constuctor creates and stores two variables, both of which are needed for OgreHAPI to work. *mHapticsDevice* holds a pointer to a *HAPI::AnyHapticsDevice*, which is any haptics device supported by the HAPI package. If, initially, a haptics device is not turned on or even plugged in, OgreHAPI will automatically catch the first haptics device it supports and use that when the application is running. *mSpace* holds a pointer to the haptic space where all haptic objects will have to be added to if they are to be haptically rendered.

```
WhatsInTheBoxApplication::
  WhatsInTheBoxApplication(void)
  {mHapticsDevice =
    new HAPI::AnyHapticsDevice();
   mSpace =
     new OgreHAPI::Space(mHapticsDevice);}
```

4.3. Step Three: Adding Haptic Objects.

```
objects[0] = new OgreHAPI::
  Objects::MeshObject(ent->getMesh(),
  new HAPI::
    FrictionSurface(1.35,0,0.2,0.2),0.1);

objects[1] = new OgreHAPI::
  Objects::Sphere(0.0375,
  new HAPI::
    FrictionSurface(1.35,0,0.2,0.2));
```

The first object uses the OGRE entities mesh information to build the haptic object, whereas the second uses a primitive shape. It is recommended to use primitive shapes where possible as they are processed significantly faster than mesh objects. The *MeshObject* constructor takes an *Ogre::MeshPtr* object, a *HAPI::HapticSurfaceObject* and then the radius of the bounding sphere (at the moment this must be the size in the haptic world not OGRE world, i.e. the haptic world without the scale). The Sphere constructor takes the radius of the sphere (again in without the scale applied to it) and a *HAPI::HapticSurfaceObject*.

Finally the *OgreHAPI::Object* is attached to an *Ogre::SceneNode*. This will ensure that the haptic object will follow the position and orientation of the specified scene node. Also the haptic object needs to be added to the haptic space so that it will be rendered:

```
objects[0]->attachToSceneNode(nodes[0]);
mSpace->addObject(objects[0]);
```

5. Evaluation Study

A user study of the OgreHAPI environment was conducted to evaluate the degree of cohesion between the haptic technology and the virtual environment toolkit. A within-subject design was used for the study with all the participants receiving the same treatment (see Section 5.1). Six participants volunteered for the study. All participants were male and within the age range of 21-32 years. All participants used a computer on a daily basis, both at work and at home and regularly played computer games. Approximately 83% were right handed and none were ambidextrous. Ethic permission for the study was granted by the Computer Science Ethics Committee (Durham University).

The evaluations were carried out in an empty room, to reduce participant distractions, and were conducted by the first author. The equipment consisted of a Pentium 4 CPU 2.4GHz PC with 1GB RAM running Windows XP (SP3), a Dimensional Technologies display (in 2D mode), a standard mouse and keyboard, and a PHANTOM Omni haptic device (see Figures 2).

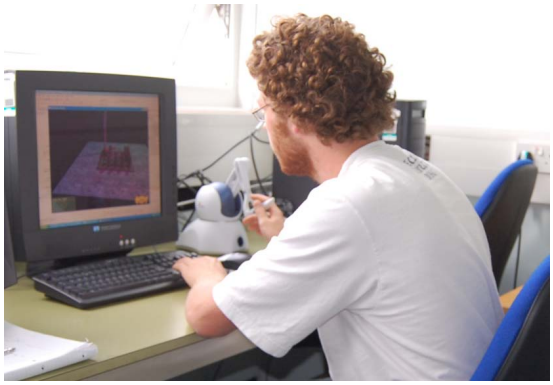

Figure 2: *User study equipment and conditions.*

5.1. Evaluation Procedure

Demographic, computer usage and haptic technology experience information was gathered in a pre-session questionnaire. A consent form was also signed by all participants. The evaluation study involved two tasks in applications implemented with OGRE graphics and HAPI haptics support, integrated through OgreHAPI.

What's In The Box?: Participants were presented with an opaque screen which obscured a haptically rendered object. Participants were asked to feel three objects (Objects A, B, and C - see Figure 3). Each of the participants had a different permutation of the ordering to balance the experiment, e.g. participant 1 had objects in order ABC, participant 2 has order ACB, participant 3 had order BAC and so on. After having the opportunity to feel the haptic rendering, the participants were asked to write down what they thought the object was, or to give a brief description. After participants had completed descriptions of the three objects they were then shown images of all the objects and were asked to rate the difficulty of identifying the different objects ([Hard] *1 2 3 4* [Easy)]) and to rate how realistically the shapes were rendered ([Unrealistic] *1 2 3 4* [Realistic]).

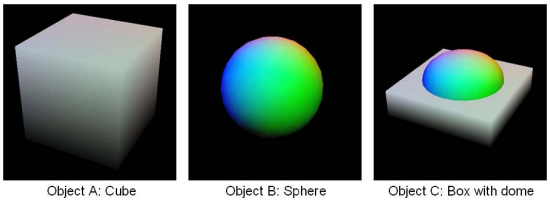

Figure 3: *What's in the Box objects.*

Bar Skittles: The second part of the evaluation required the participants to play two rounds of a *Bar Skittles* game. The game presented a first-person view of a 3D virtual environment (see Figure 4). The object of the game is to knock over all the skittles by repeated pushing the ball, which is attached to a pole, with an in-game avatar controlled by the haptic device. The environment provided realistic physics for the ball and pin interactions. The physics in this task were provided by *OgreNewt* which is a library that wraps the Newton Game Dynamics physics SDK (http://www.newtondynamics.com/) into OGRE. Timing metrics were collected for both rounds of the game.

Participants then completed a questionnaire of 4-point Likert scale questions on the difficultly of playing the game and the perceived realism of the game. Space was also provided on the questionnaire for short comments.

6. Results

All the participant completed both tasks. The results of the What's in the Box task can be seen in Figure 5. There were

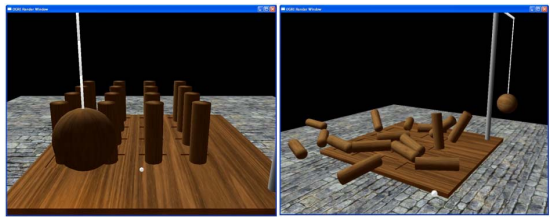

Figure 4: *The Bar Skittles game environment. The white dot is the user's avatar in the environment.*

high levels of correct answers for all three objects. This was surprising for *Object C* as this object had an unusual shape. That the majority of participants could identify *Object C* without visual feedback indicates a good level of accuracy in the haptic integration.

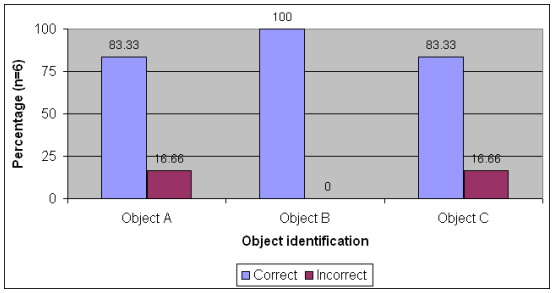

Figure 5: *Answers from the What's in the Box task.*

The timings from the Bar Skittles task can be seen in Figure 6. All participants improved on their timings between rounds. Therefore the haptic renderings, graphical renderings and the implemented physics, were not interfering with expected skills improvement through practise. This is encouraging in terms of the integration of the haptic and graphical feedback in the environment.

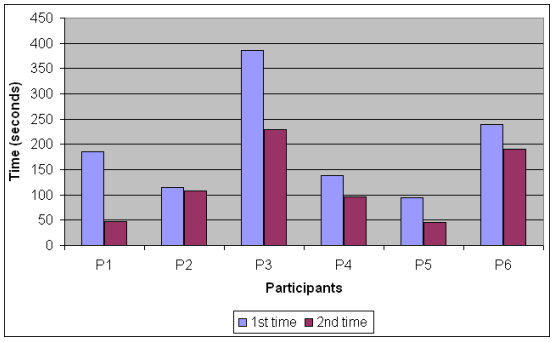

Figure 6: *Timings from the Bar Skittles task.*

Figure 7 shows the results of the questions on difficulty and realism over the two task environments. There were high

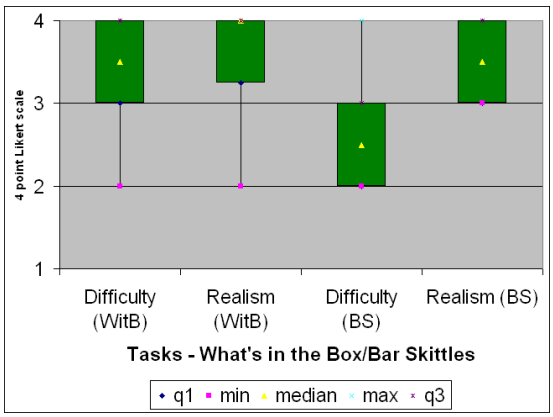

Figure 7: *Boxplots of participant's answers to difficulty and realism questions for both tasks.*

values for both task environments. The Bar Skittles environment was found to be more difficult and participant comments indicated issues with the in-game avatar, i.e. the haptic device representation. This was a small sphere and participants noted that it was "hard to tell where the avatar was in the 3D world" and that the avatar "kept slipping off of the ball". One suggestion was to provide a hand avatar to grab and swing the ball, in contrast to the current method of pushing the ball. No participants suggested improving the haptic interaction.

7. Conclusions

There is increasing demand for haptic, or touch-based, interaction in virtual envinronments. Although many haptic devices come with APIs to enable the development of haptic-based applications, many do not provide the same level of graphical support available in virtual reality or game technology toolkits. It is important for the layer between the haptic device and the graphics application to be extensible and customizable [DP07]. This paper has described a middleware library of classes to support the integration of a device independent haptic API with an existing virtual environment engine. This work specifically provides an extendable framework for linking OGRE and the HAPI API. More generally it provides an example framework for linking virtual environment engines to a device independent haptic API.

Proof-of-concept has been demonstrated with a user study where a high level of performance and acceptance of the haptic integration was found. However, as middleware is created for developers, to evaluate the flexibility and ease of use of the proposed framework, a next step is to ask developers to use the library, and its functions, to integrate a haptic device into a virtual environment. This is ongoing work.

The OgreHAPI library, class documentation and two tutorials for building OgreHAPI applications are available online at http://www.dur.ac.uk/shamus.smith/ogrehapi/.

8. Acknowledgements

The authors would like to thank the evaluation study participants. This work was funded in part by the Nuffield Foundation (Grant URB/35744).

References

[BKLP05] BOWMAN D. A., KRUIJFF E., LAVIOLA JR. J. J., POUPYREV I.: *3D User interfaces: Theory and Practise*. Addison Wesley, USA, 2005.

[BS02] BIGGS S. J., SRINIVASAN M. A.: Haptic interfaces. In *Handbook of Virtual Environments*, Stanney K. M., (Ed.). Lawrence Erlbaum Associates, New Jersey, 2002, pp. 93–115.

[DP07] DE PASCALE M., PRATTICHIZZO D.: The Haptik library. *IEEE Robotics & Automation Magazine* (December 2007), 64–75.

[HCK*01] HUBBOLD R., COOK J., KEATES M., GIBSON S., HOWARD T., MURTA A., WEST A., PETTIFER S.: GNU/MAVERIK : A micro-kernel for large-scale virtual environments. *Presence: Teleoperators and Virtual Environments 10* (February 2001), 22–34.

[Kal93] KALAWSKY R. S.: *The Science of Virtual Reality and Virtual Environments*. Addison-Wesley, 1993.

[LaV08] LAVIOLA JR. J. J.: Bringing VR and spatial 3D interaction to the masses through video games. *IEEE Computer Graphics and Applications* (September/October 2008), 10–15.

[PM04] PAVA G., MACLEAN K. E.: Real time platform middleware for transparent prototyping of haptic applications. *International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems* (2004), 383–390.

[Smi07] SMITH S. P.: Exploring the specification of haptic interaction. In *Interactive Systems: Design, Specification and Verification* (2007), Doherty G., Blandford A., (Eds.), vol. 4322 of *LNCS*, Springer, pp. 171–184.

[SMK98] STANNEY K. M., MOURANT R. R., KENNEDY R. S.: Human factors issues in virtual environments: A review of the literature. *Presence: Teleoperators and Virtual Environments 7*, 4 (August 1998), 327–352.

[WB06] WALL S., BREWSTER S.: Editorial: design of haptic user-interfaces and applications. *Virtual Reality 9*, 2-3 (2006), 95–96.