Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS

Petr Kadleček

A Practical Survey of Haptic APIs

Department of Software and Computer Science Education

Thesis supervisor: Mgr. Petr Kmoch
Study program: Computer Science, Programming

2010

I would like to thank my supervisor, Mgr. Petr Kmoch, for his support and advice throughout the survey, development of accompanying applications and the thesis. He provided me with valuable information, insight into haptics and guided me through the work.

Contents

Název práce: Rešerše haptických API
Autor: Petr Kadleček
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: Mgr. Petr Kmoch
e-mail vedoucího: petr.kmoch@mff.cuni.cz

Abstrakt: Haptická zařízení mají velký potenciál v oblastech jako je lékařství, inženýrství, či pomoc zrakově postiženým lidem. Masivní rozšíření a podpora umožnila vznik mnoha nástrojů a knihoven pro programování aplikací s podporou haptiky. Tato práce představuje a analyzuje sadu rozhraní pro haptické programování, které převážně podporují zařízení Novint Falcon. Součástí práce je také sada testovacích programů ukazující základní použití těchto rozhraní. Práce se dále zaměřuje na knihovnu CHAI 3D, pro niž byla vytvořena rozsáhlejší testovací aplikace a implementována podpora multiplatformního ovladače libnifalcon.

Klíčová slova: haptické zařízení, haptická API, CHAI 3D, libnifalcon, Novint Falcon

Title: A Practical Survey of Haptic APIs
Author: Petr Kadleček
Department: Department of software and computer science education
Supervisor: Mgr. Petr Kmoch
Supervisor's e-mail address: petr.kmoch@mff.cuni.cz

Abstract: Haptic devices have a great potential in medical fields, engineering and help for visually impaired people. Massive distribution and support have made it possible to create many tools and libraries for programming applications with support of haptics. This thesis presents and analyzes APIs for haptic programming which mostly support a Novint Falcon device. A part of the thesis is a set of testing applications illustrating a basic use of these APIs. The survey then focuses on a CHAI 3D API. Larger demonstrating application is created and a libnifalcon cross-platform driver is implemented to the CHAI 3D.

Keywords: haptic device, haptic API, CHAI 3D, libnifalcon, Novint Falcon

Chapter 1

Introduction

Display and audio engineering have developed rapidly over the last few decades and brought realistic reproduction to our vision and hearing senses. Haptic technology makes it possible to use the sense of touch with computers. Progress of this technology enables humans to create an immersive virtual reality.

Commercial haptic device available for consumers was released in the year 2007 and there are dozens of accessible application programming interfaces, development kits and libraries at different abstraction layers.

There are APIs more suitable for fast prototyping in the early stages of a software development and APIs suitable for efficient control of haptic device for applications requiring accurate real-time responses. Choosing a suitable haptic API becomes an integral part of a project supporting haptic technology.

1.1 The goals of the thesis

The main goals of the Practical Survey of Haptic APIs are:

- present current haptic APIs for controlling haptic devices with a focus on open source, cross platform APIs supporting the Novint Falcon device

- analyze these APIs and create basic testing applications to give a thorough review for developers starting with haptic programming

- examine CHAI 3D library in a more detail and create a larger project using CHAI 3D
- consider an implementation of libnifalcon cross-platform driver into CHAI 3D and test its performance

1.2 Structure of the thesis

The thesis contains 6 chapters:

- Chapter 1 gives a general introduction and presents goals of the thesis
- Chapter 2 introduces a haptic technology, haptic devices and practical use of haptics.
- Chapter 3 examines haptic API abstraction layers and gives a review of haptic APIs with a short conclusion and recommendations. The overall summary of licensing, development state and other specification is at the end of the chapter.
- Chapter 4 describes an implementation of libnifalcon driver into CHAI 3D API and presents result of performance testing benchmark.
- Chapter 5 provides an overview of Haptic API suite - a package of testing applications along with precompiled and modified APIs
- Chapter 6 concludes the work and describes possible future upgrades and extensions to the survey

An italic font in the text specifies a fragment of source code such as a *class* name, *method* or a *function*. A verbatim font is used to emphasize and preserve format of the source code such as:

```
hello_world :- write('Hello World!').
```

Chapter 2

Haptic device

2.1 Human-Computer Interaction

Most users communicate with computers using a mouse and a keyboard. This kind of interaction is 30 years old and does not benefit from many senses humans possess. Working with only the mouse and the keyboard demands a great cognitive load which limits the use of computers. Human-Computer interaction (HCI) is a field of study concerned with principles of how humans communicate with computers and computers communicate with humans in any possible way.

On one hand, the mouse and the keyboard are cumbersome with respect to overall control mechanisms. On the other hand, a man can't hold an object or even his hand in the air for the whole day. When users work with the mouse properly they should have their hands relaxed. Beside this, development of new HCI devices that could compete with the old ones is very limited by existing long-established interface. Therefore, new devices are often applied in special-purpose facilities or in accessories for a specific use (e.g. mobile phones with touch screen, interactive head-up displays in vehicles and airplanes, Microsoft Kinect, ...).

A haptic device allows humans to send information to the computer by moving a part of the body (often with a hand using a small sphere shaped object called grip, a pen shaped object or gloves) and receive information from the computer by a force feedback generated by the device.

Although the sense of touch is not as acute as hearing, its accuracy is somewhere in between sight and hearing. Humans need approximately 500 Hz to 1000 Hz frequency of a haptic feedback to achieve smooth force perception, according to [4].

While force feedback gives a sense of force or generally a kinesthetic feel, tactile sensing is used when one wants to feel pressure, heat or fine textures (and any other sensation felt by the skin). Technology prototypes using both kinesthetic and tactile feedback have been released.

2.2 Degrees of freedom and variability of current haptic devices

Haptic devices can be generally divided by the dimension of an orientation ability called degrees of freedom (DOF). That is basically translation (3-DOF) and translation combined with rotation (6-DOF). A typical example is a movable grip for 3-DOF devices (e.g. Novint Falcon) and a pen on a pivot with the ability to rotate and translate both in all three dimensions (Sensable Phantom Omni). There are also 6/3-DOF devices that combine 6-DOF positioning and 3-DOF force feedback. 7-DOF devices have a scissors snap-on, a thumb-pad or any other extra grip.

A list of haptic hardware [1]:

2-DOF devices
: Quanser planar pantograph

3-DOF devices
: ForceDimension omega.3, ForceDimension Delta.3, Novint Falcon, Quanser Mirage model Haptic Wand

5-DOF devices
: Immersion Laparoscopic Surgical Workstation, Quanser twin-pantograph ”Haptic Wand”

6/3-DOF devices
: Sensable Phantom Omni, ForceDimension Omega.6

6-DOF devices
: ForceDimension Delta.6, Sensable Phantom Premium, Sensable Phantom 3.0, Haption Virtuose

[1]based on `http://www.bracina.com/haptichardware.html`

7-DOF devices
: Sensable Phantom with scissors snap-on and effector, Force Dimension Omega.7

A selection of common comparable properties which can be found in technical specifications of haptic devices:

Workspace
: Specifies a maximal reach of a touch tool (often measured in inches) and maximal rotation abilities if appropriate.

Position resolution
: Resolution of a touch tool position measured in dots per inch (DPI).

Maximal force
: Maximal force can be specified in newton unit or as a force capability in kilograms or pounds.

Stiffness
: Stiffness of a haptic device measured in newtons per metre.

2.3 Practical applications of haptic devices

One of the most valuable applications of haptic devices is in medicine. Extremely accurate position resolution and force feedback is necessary for surgical operations or simulations while using teleoperation of medical tools (e.g. laparoscopy). A haptic device can be also used as a virtual examination tool for medical data of a patient such as computed tomography (CT) scans.

Haptic devices are also valued as assistive technology for visually impaired or blind people. Haptic technology enables them to use the sense of touch to retrieve information such as depth or the contour of an object from a computer. A precise tactile feedback will propose a dynamic surface for the Braille system.

There are numerous applications of haptic devices in engineering. A few inspirational examples can be found in a selection of demos from the latest Eurohaptics [2] conference as of writing the thesis: Haptesha: A Collaborative Multi-User Haptic Workspace, TexturePad: Realistic Rendering of

[2]`http://www.eurohaptics2010.org/hod.shtml`

Haptic Textures, Haptic gas pedal capable of recording proprioceptive feedback parameters, New design of a touchpad device with tactile feedback, Electro-tactile Display with Real-time Feedback.

Other applications can be found in military, painting, CAD systems and gaming.

2.4 Novint Falcon

The survey will primarily focus on the Novint Falcon device, as it was the only haptic device available throughout the creation of this thesis.

Novint Falcon is a first low-cost commercial haptic device primarily intended as a game controller. It is a 3-DOF device with a removable grip which can be replaced with a special purpose grip such as a pistol grip for games. The workspace of Novint Falcon is 4 inches in all three dimensions, position resolution starts at 400 DPI and it can produce force sensation of approx. 8 newtons.

The device has 3 encoders that read position of arms, 4 buttons (sometimes called switches) and LED identification of current device state as shown in Figure 2.1.

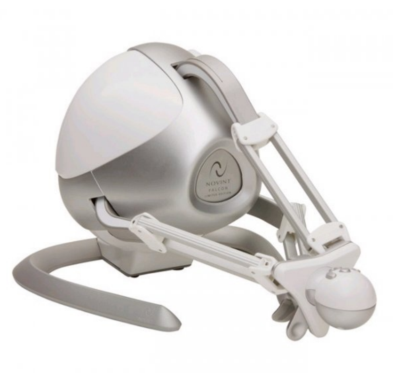


Figure 2.1: Novint Falcon haptic device

Novint Falcon is now officially supported on Microsoft Windows XP SP2 and Vista operating systems only. There have been some issues regarding installation of Novint Falcon drivers on Microsoft Windows 7 64 Bit found on community website [3].

A Falcon Test utility provides a diagnostic information about all motors, encoders, buttons, front LED diagnostic and calibration of arms called homing by extending all three arms as far out as it is possible. A software development kit called HDAL (Haptic Device Abstraction Layer) is available to download at Novint website [4].

[3]`http://www.falconarmy.com`
[4]`http://home.novint.com/products/sdk.php`

Chapter 3

Haptic APIs

3.1 Abstraction layers of haptic APIs

There are various methods of implementing haptic device control into an application ranging from the lowest driver layer to the highest scene graph layer. The most important decision a software architect has to take into account is a choice of the particular abstraction layer at which the rest of the application communicates with haptics.

3.1.1 Driver layer

The lowest layer at which the programmer can communicate with the device is a driver of the operating system. At this layer the driver receives raw data through a serial bus (e.g. USB, IEEE 1394) from encoders that has to be processed with kinematics algorithms to get the data that corresponds to a three-dimensional vector of the haptic tool position in cartesian coordinates. Manual initialization, opening and closing communication with the device or an inverse kinematics algorithm which computes force data in the application and sends it to the device to compute angles at haptic device joints is also essential. To preserve a smooth haptic response thread handling has to be done. For this reason, an extra haptic thread which calculates physics in the application is necessary.

The driver layer provides the fastest and the most precise response but demands a great effort to get the device working. Support of any other haptic device that has no compatible communication protocol means rewriting a lot of source code.

Manufacturers of haptic devices often provide optimized and well documented drivers in the C or C++ programming language. There are also open source and cross platform drivers that can provide support in officially unsupported operating systems such as Linux or Mac OS.

Use of a higher abstraction layer API that uses standardly available drivers won't be possible unless the support of an extra driver is added.

A driver layer is often used for very specific real-time applications where immediate response is vital.

3.1.2 Low-level API

While the driver layer communicates in raw data, a low-level API hides kinematics algorithm implementation from the programmer and allows developers to work directly with position, rotation and force vectors in the application.

Many low-level APIs works as a common interface for different drivers which is very helpful when supporting a lot of haptic devices. A device handler is then used for getting information on haptic devices available on the current machine.

A particular set of functions and capabilities associated with a low-level API is not strictly defined. There are low-level APIs that provides a lot more functionality than the haptic device handler.

Haptic rendering

One of the most important algorithmic problem associated with haptics is computation of interactions between the haptic tool and virtual objects. Creating a convincing force reaction at the edge of a complex object becomes a nontrivial task that is dependent on data representation. Such a technique of haptic interaction processing in the virtual scene is called haptic rendering (or haptic display). As in graphic rendering, where the image is composed from a model based on a virtual camera position, the process of haptic rendering returns a force on the basis of a model with which the haptic tool interacts. Creating a good haptic rendering algorithm is a struggle to maintain realistic force feedback without using cumbersome computations which raise memory and CPU requirements.

A god-object method proposed by Zilles and Salisbury [1] is one of the most implemented methods of haptic rendering. The god-object itself is a proxy model of the haptic tool (a virtual haptic interface) within the virtual world that helps with the returning force calculations between servo cycles of the haptic device. Another well known method is described in Ruspini, Kolarov and Khatib [2].

Low-level APIs may support one or more haptic rendering methods in the virtual haptic world defined in the low-level API.

Besides pure haptic rendering, low-level API may provide a variety of haptic effects such as a spring effect, magnet effect or any other surface effect.

A low-level API is often a good choice when good haptic performance is needed while using one's own graphics rendering method.

3.1.3 Scene graph API

The graphical and haptical data representation of a model may be very similar or sometimes even identical. Integration of graphics and haptics into one API is therefore reasonable.

A scene graph haptic API often uses a tree structure of objects in the virtual world with a specific root node such as a world node. It is possible to apply graphical and haptical properties to an object and set the specific property recursively to its children objects.

A high-level API often includes low-level APIs for haptics, graphics, physics and audio processing. It provides all the features of low-level APIs and even more by combining them together. Haptic and graphic rendering is essential in the scene graph API oriented on haptics.

The concept of combining low-level APIs into one often creates many drawbacks which the high-level scene graph API implementation may or may not hide from the programmer. Difficulties connected with such a combination of different APIs may result in a thorough problem analysis that may not even be solved with a feasible effort because the API itself may be proprietary and authors may not support the API any more.

A scene graph haptic API is the best choice for prototyping an application when the speed of development is crucial and performance is not a priority. Support of a scripting language or standard file format representation of a scene helps even more with rapid development.

3.2 CHAI 3D set of libraries

CHAI 3D [3] is a scene graph API written in the C++ programming language with aim to create a modular, open source and cross platform haptic API with a wide support of different haptic devices. CHAI 3D is licensed under GNU General Public License (GPL) version 2 [1] but offers even a Professional Edition License. The main reason to create CHAI 3D was that all available APIs developed by manufacturers of haptic devices were proprietary and supported only the one specific device or a group of devices from the manufacturer.

The scene graph capabilities of CHAI 3D mainly focus on haptics combined with graphics. It does not include any extra visual or sound effects but it does propose lightweight and compact functionality. CHAI 3D is definitely not the API with tons of functions ready for the implementation of sophisticated applications. It is rather the API for academic and research use where the extra functionality can be added.

Though the API manual or tutorials do not yet exist, the source code is very well documented and is very easy to read and scan through. The reference guide generated by a Doxygen documentation system [2] could serve as a quick guide over the source code but it is not a comprehensive source of learning CHAI 3D. Authors of CHAI 3D recommend to learn by the examples in packages for different platforms. This method gives the learner a decent overview of the API but does not allow to fully understand some fundamental characteristics of the API which makes the learner read part of the API source code eventually.

What makes CHAI 3D source code easy to read is a coding convention definition [3] which defines a few aspects of coding that should be obeyed in order to write code of the API. One of the most visible and helpful aspect for a learner is separation of classes, class members, function parameters and local variables by a preceding letter (e.g. *cGenericObject* for a class, *m_parent* for a class member and *a_affectChildren* for a function parametr).

The CHAI 3D library is split into several modules and class groups that provide specific tasks: Devices, Graphics, Math, Widgets, Scenegraph, Haptic Tools, Haptic Effects, Force Rendering, Collision Detection, Timers, Files, Extras, Viewports, GEL Dynamics Engine and ODE Dynamics Engine.

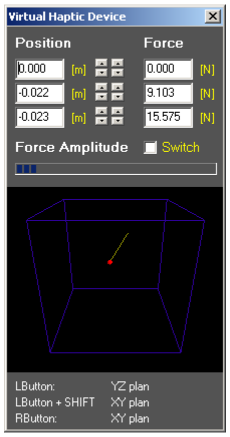
[1] `http://www.gnu.org/licenses/gpl.html`

[2] `http://www.stack.nl/~dimitri/doxygen/`

[3] `http://www.chai3d.org/coding.html`

3.2.1 Devices

CHAI 3D supports devices from Force Dimension, Novint Technologies, MPB technologies and Sensable technologies. A virtual device for Microsoft Windows operating system was specifically developed for an experimental haptic lecture [4]. It provides a graphical representation of workspace, the haptic tool and generated force as a vector as shown in figure 3.1 and allows the user to move the haptic tool using computer mouse and keyboard.



Position		Force	
0.000	[m]	0.000	[N]
-0.022	[m]	9.103	[N]
-0.023	[m]	15.575	[N]

LButton:	YZ plan
LButton + SHIFT	XY plan
RButton:	XY plan

Figure 3.1: CHAI 3D virtual device

Adding support of a new haptic device is simple thanks to the modularity and well documented source code. Support of libnifalcon - a Novint Falcon cross platform driver - was added to CHAI 3D as part of this thesis (more on that in chapter 4).

Low-level use of API

Though the CHAI 3D library is a scene graph API, use of CHAI 3D as a low-level communication layer is convenient. CHAI 3D provides support of many devices and an easy to use device handler *cHapticDeviceHandler*.

[4] `http://cs277.standford.edu`

Every device is then treated as a generic haptic device *cGenericHapticDevice* with basic ability to get a position, set a force, device communication opening, initialization and closing.

This method had a small drawback in the older version of CHAI 3D where it was not possible to link the CHAI 3D library without OpenGL libraries due to very close integration of OpenGL methods which made project code dirty especially when using Microsoft DirectX for graphics rendering.

3.2.2 Scene graph

A scene graph of CHAI 3D contains standard shapes, meshes, virtual cameras and lights.

Objects

The main unit of all objects in the scene graph is a *cGenericObject* class which inherits from a general abstract type *cGenericType*. The generic object creates a tree structure of objects using a standard template vector class of children objects in a *m_children* member. All methods for object modification or property setting allow propagation to children by setting an optional function parameter *a_affectChildren*, which is by default set to false. CHAI 3D scene graph has one root node class for every object in the scene called *cWorld*. This class is essential for further communication with graphics and haptics.

The API contains only three standard object shapes (two implicit surface objects) :

- a sphere (*cShapeSphere*) defined by a radius
- a torus (*cShapeTorus*) defined by an inside and an outside radius
- a line (*cShapeLine*) defined by two points as three-dimensional vectors

Adding such an object into a scene graph means calling a constructor of the specified object with appropriate properties (e.g. radius of the sphere) and adding it as child (using an *addChild* method) into another object that is already in the scene, or directly to the world root node. Graphic and haptic rendering is then performed from the root node recursively to children.

Beside standard shapes implemented in CHAI 3D API, it is possible to load complex meshes of two file formats:

- .OBJ - geometry file format from Wavefront Technologies[5]
- .3DS - 3D Studio file format from AutoDesk[6]

Both file formats contain a list of vertices with their relative positions, a list of polygons, material and texture references, texture mapping coordinates and optionally normals (manual calculation of normals is necessary in 3DS file format).

Camera and lights

A virtual camera in the scene initiates the graphic rendering process by calling a *renderView* method with appropriate function parameters of window width, window height and image index identifying optional stereo rendering frame. The virtual camera then renders all objects in the parent world into which it was added.

The camera is basically a wrapper for the *gluLookAt* (part of the OpenGL Utility Library) viewing transformation function with eye position, center position (a look at position) and an up vector. A *cCamera* class is written in a modular way so that creating a custom viewing transformation function that alters the viewing matrix is a matter of adding a new class to the scene graph API which alters the OpenGL viewing matrix and calls *renderSceneGraph* method of the parent world.

As stated in the reference guide, the camera looks down the negative x-axis which is the standard convention in general robotics. This coordinate system convention may be confusing especially while working with complex meshes edited in applications with a different coordinate system.

The lighting system was also adapted from the OpenGL library and provides basic OpenGL functionalities wrapped in a *cLight* class. Parent world manages a light source list on its own so that the programmer only sets the properties and adds the light directly to the world or attaches the light to the camera by adding it as a child of the camera instance.

3.2.3 Haptic tool

The scene graph representation of a haptic device is called a tool. An abstract class defining all tools in the scene graph is *cGenericTool*. The only specific

[5] `http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj`

[6] `http://www.martinreddy.net/gfx/3d/3DS.spec`

tool that CHAI 3D provides at this time is a 3-DOF tool identified as a *cGeneric3dofPointer*. 6-DOF force rendering algorithms are not supported.

The generic tool is also a generic object which means that the tool has its position, rotation and all other object properties. The tool itself needs only a pointer to the haptic device from a device handler. It manages all the initialization automatically by calling a *start* method. A *stop* method does the opposite.

The default device mesh of the generic 3-DOF pointer displays the tool as a sphere. The god-object algorithm mentioned in section 3.1.2 is used for the haptic force rendering for which there are two meshes representing the tool:

- a device mesh (*m_deviceMesh*) which represents the real current position of the haptic device touch tool
- a proxy mesh (*m_proxyMesh*) which represents a model of the haptic interface in the virtual environment

The force model is also defined as the abstract model (with a generic class *cGenericPointForceAlgo*) split into *cProxyPointForceAlgo* and *cPotentialFieldForceAlgo* classes. The *cProxyPointForceAlgo* class implements the God-object method with collision detection and *cPotentialFieldForceAlgo* class process local interaction relating to haptic effects. Interaction event structures are defined as *cInteractionEvent*, *cInteractionRecorder* and *cInteractionSettings*.

An overall force contains assigned local haptic effects and interaction forces computed on the base of haptic device properties (e.g. stiffness), a position relative to an interaction projected point on the interacting object surface and a best new position of the proxy model in the proxy point force algorithm. Interaction detection is not always precise especially in complex meshes and the proxy model gets sometimes stuck and generates excessive force.

The tool works in a workspace set by a radius. It is possible to change the radius and position of the workspace and its rotation relative to the scene. The tool is often attached to the camera so that the workspace corresponds to the view of the camera.

3.2.4 Haptic effects

The CHAI 3D scene graph provides a set of haptic effects that can be assigned to objects. These effects are computed using a local interaction *computeLocalInteraction* method of each implicit surface object. The mesh or any other complex object without overridden *computeLocalInteraction* method is not able to compute haptic effects because there's no way how to compute an interaction projected point from a generic object algorithm. A temporary mesh local interaction computation was implemented in the CHAI 3D testing application described in a section 5.2.

Haptic effects with the base abstract class *cGenericEffect* in the API are as follows:

- Magnetic model effect *cEffectMagnet* provides a magnetic field effect near the object
- Stick-slip effect *cEffectStickSlip* provides an effect of sliding one object on another with sticking caused by friction (e.g. rubber on a desk)
- Surface effect *cEffectSurface* provides a basic surface effect of a tool pushing against the object
- Vibrations effect *cEffectVibrations* provides an effect of a vibration with a specific frequency and amplitude
- Viscosity effect *cEffectViscosity* provides an effect of a tool moving through a fluid

All effects are very sensitive to a good setting of properties such as a maximal stiffness of the haptic device. A relatively small change of effect properties can make a great difference in the effect perception and sometimes even a different driver may result in a different effect behavior.

3.2.5 Other classes

There are a few other auxiliary classes in the CHAI 3D API that allow the programmer to spend time on logic of the application rather than on creating (not only) system specific functionality.

Collisions

The API provides standard optimized collision detection algorithms that can be used in the scene graph. Beside a brute force collision detection algorithm *cCollisionBrute*, CHAI 3D offers an Axis-Aligned Bounding Box (AABB) tree collision detection *cCollisionAABB* and sphere tree collision detection *cCollisionSpheres*.

A collision detector is then created for every object in the scene by calling the appropriate method (e.g. *createAABBCollisionDetector*).

Graphics

There are many classes which help managing the graphics part of the scene graph: class defining a color (*cColor*), vertex (*cVertex*), OpenGL texture (*cTexture2D*), object material (*cMaterial*), etc.

Math

The *cMaths* class provides very helpful inline functions such as absolute value, linear interpolation or clamping. The *cMatrix3D* class defines a three-dimensional matrix and the *cVector3D* class defines a three-dimensional vector. There is also a class to represent rotations in quaternion form *cQuaternion* and a *cString* helper class for easier conversion between numerical and string values.

Timers

A precise timer is often needed in real-time graphic and haptic rendering to compute the time of graphics and haptics steps. A *cPrecisionClock* class provides a high-resolution timer with start, stop and reset methods. The timer uses very precise *QueryPerformanceCounter* and *QueryPerformanceFrequency* functions with resolution in the order of microseconds on a Microsoft Windows platform. The *GetTickCount* function used on other platforms has a resolution in the order of milliseconds which is sufficient in most cases.

Basic thread handling is provided in the *cThread* class so that it is possible to start a haptic thread without calling system specific functions. Adding another cross-platform thread handling library (e.g. a Boost library [7]) is rec-

[7] `http://www.boost.org/`

ommended to obtain more sophisticated mutex handling.

Widgets

Widgets offer a way of creating a two-dimensional graphical user interface on top of the three-dimensional scene. A *cBitmap* class loads only BMP and TGA image file formats using a *cImageLoader* class on all platforms. On a Microsoft Windows platform CHAI 3D uses *OleLoadPicturePath* which is able to load more file formats. Unfortunately the function in the CHAI 3D 2.0.0 version has wrongly implemented working directory location and doesn't work properly. A fix can be found in the Haptic API suite (described in chapter 5) version of CHAI 3D. Nevertheless, even an OLE function doesn't provide a way how to load a PNG image file with alpha-channel support. Libpng [8] library support in CHAI 3D was also added in the Haptic API suite.

A *cLabel* class renders a two-dimensional text with specified properties on the screen using a *cFont* class.

3.2.6 ODE module

The CHAI 3D library does not implement its own rigid body dynamics simulation. There is, however, a module that connects the CHAI 3D scene graph with the Open Dynamics Engine [9] (ODE) library.

Communication of CHAI 3D and ODE is handled by *cODE*, *cODEWorld* and *cODEGenericBody* classes. The API contains precompiled ODE libraries for both dynamic and static linking with double precision. Preprocessors definitions need to be set correctly in order to run an application properly without runtime errors. It is necessary to tell the ODE library to use double precision by adding a preprocessor symbol **dDouble**. Such an information would be very helpful in the API manual or at least as a comment in the source code of examples of the ODE module for programmers without knowledge of the ODE library.

Every object in the ODE simulation has to be added to a specific ODE world. Such an object is defined as an ODE generic body with properties of physical simulation and an image model of the scene graph. The ODE world is a generic object which behaves as a child in the standard parent world

[8] `http://www.libpng.org/pub/png/libpng.html`
[9] `http://www.ode.org/`

but has a list of bodies instead of a list of children. This behavior affects all recursive algorithms in the scene graph. For instance, it is therefore not possible to assign a haptic effect to an object in the ODE simulation. A fix of this behavior can be found in the Haptic API suite (described in chapter 5).

The ODE module enables creation of a dynamic box, sphere, capsule and a mesh from an assigned image model. Static planes are also available. A global gravity can be set as a three-dimensional vector describing a force. Calling an ODE world *updateDynamics* method with a step time function parameter updates the simulation.

Though the implementation of dynamics into the scene graph is simple, a programmer still has to work with the ODE world as a separate world and encounters a lot of disadvantages when using recursive scene graph algorithms.

3.2.7 GEL module

The haptic technology utilizes an implementation of a deformable body simulation more than any other technology. CHAI 3D provides a module to create such deformable objects in the scene graph which uses the GEL dynamics engine developed at Standford University.

As in the ODE module, the GEL module is implemented as a separate world (*cGELWorld*) of deformable objects. The main idea behind the deformation is a skeleton model made of nodes (*cGELSkeletonNode*) and links (*cGELSkeletonLink*) between them. Nodes are represented as spheres with a given radius and mass connected with elastic links with spring physics defined by elongation, flexion and torsion properties. Every node has its physical properties (linear damping, angular damping, gravity field definition) and provides methods to control force and torque.

The GEL module provides a simple way to add deformable objects to the scene graph, but integration of the GEL dynamics engine in the lower layer of the scene graph with automated skeleton modeling would considerably enhance the high level use of CHAI 3D.

3.2.8 BASS module

Another external module of the CHAI 3D API is a BASS module. BASS [10] is a library providing functions to manage audio samples, streams and recording with a large support of many audio formats. The module itself has no specific integration to the scene graph and it is up to the programmer to read the BASS documentation and use BASS functions directly.

3.3 Novint HDAL SDK

Novint Haptic Device Abstraction Layer (HDAL) is a software development kit specifically developed for the Novint Falcon haptic interface device. The HDAL SDK provides a low-level interface to haptic device for applications written in the C/C++ programming language on Microsoft Windows (XP or newer) operating systems. A noncommercial licence agreement [11] grants an own personal purpose and noncommercial use of SDK with limitations of any form of reverse engineering and discovering principles of operation of the SDK.

The HDAL SDK contains precompiled dynamically and statically linkable libraries, C include files, examples, utilities, documentation and a reference guide. The documentation called HDAL Programmer's Guide familiarizes a programmer with haptics programming, Visual C++ integrated development environment (IDE) settings of HDAL and the HDAL use in an application.

The HDAL interface provides basic low-level API functions to get position and send forces by using callback functions. There's no need to manage an extra thread for a haptic device in the application because the HDAL runs its own thread hidden from the programmer. A data from the haptic device can be obtained in two different ways:

- blocking servo loop callback which stops the thread where the function was called and reads the data at the frequency of 1 KHz
- non-blocking servo loop callback which works with the latest received data from the haptic device

[10] `http://www.un4seen.com/`

[11] Please read the whole licence agreement in order to use the SDK

HDAL also manages the initialization of the device with *hdlInitNamedDevice*, *hdlStart* and *hdlStop* methods. The device configuration can be specified in a special INI file HDAL.INI which contains driver DLL settings, position and offset scaling or logging level. The INI file specification can be found in the HDAL Programmer's Guide.

Higher layer utility functions of HDAL provide mapping of the haptic workspace to the application workspace (*hdluGenerateHapticToAppWorkspaceTransform*) and a high precision timer function (*hdluGetSystemTime*).

The HDAL SDK is mostly used as a system specific low-layer interface for the Novint Falcon device included in higher level APIs. Because the source code is not available, its purpose of a device independent library is at this time limited to only one device. If the programmer wants to run an application on the Microsoft Windows operating systems and has no intention to use any other devices, then HDAL is a good choice.

3.4 JTouchToolkit API

The accessibility of haptic devices and the high performance of computers in last years have caused major changes in haptics programming. Altough the most used programming language in real-time haptics is still C or C++, new ways of haptic programming emerge.

JTouchToolkit [12] is an open source haptic API written in the Java programming language, licensed under the GNU GPL v2 license and developed by User-Lab [13]. The aim of the project is to create a very easy to use API in Java environment with support of various haptic devices (currently supported are SensAble devices and Novint Falcon).

There are many drawbacks in the JTouchToolkit API which may discourage a lot of programmers:

- the development state of the API has stalled two years ago with 2.0 beta version released
- there is no practically usable documentation, nor examples of use
- the Novint Falcon device wrapper can be used only on Microsoft Windows operating systems

[12] `https://jtouchtoolkit.dev.java.net/`

[13] `http://www.user-lab.com/`

The JTouchToolkit API is basically a Java wrapper of Novint HDAL and OpenHaptics HDAPI/HLAPI. The only available documentation is generated from source code comments by the JavaDoc tool [14] which makes it complicated to start working with the API. An interesting feature of JTouchToolkit is a haptic position and motion recorder with the ability to save the recorded data to an XML file.

A description of the JTouchToolkit API example in Haptic API suite is in section 5.7.

3.5 libnifalcon library

Thanks to massive distribution of the Novint Falcon device as a new type of USB game controller, an open source alternative to the Novint Falcon driver called libnifalcon [5] was created. The libnifalcon library is written in the C++ programming language and licensed under the BSD license [15].

The driver works on Microsoft Windows, Linux and Mac OS X operating systems using two different USB access libraries:

- libusb 1.0 [16] - an open source cross platform library prefered on Linux and Mac OS X systems
- ftd2xx [17] - library used by the original Novint Falcon driver prefered on Windows systems

The driver provides communication, device firmware loading, a kinematics algorithm and managing of switches on the Falcon device grips. Authors made a deep research of all mechanical and electronic parts (motors, communication chips, DSP chips) used in the Novint Falcon and even present photos of device disassembly. The development of the library is still in progress and the main focus is now on performance of the communication.

The library substitutes not only the Novint driver and SDK, but also provides utilities and wrappers for different programming languages for an even better support of the Novint Falcon device. A *FalconCLIBase* class provides a framework for parsing and managing command line arguments of

[14]`http://java.sun.com/j2se/javadoc/`
[15]`http://www.opensource.org/licenses/bsd-license.php`
[16]`http://www.libusb.org/`
[17]`http://www.ftdichip.com/Drivers/D2XX.htm`

the application which set properties regarding the haptic device such as the device index or firmware type. There are currently two wrappers that use the *FalconDeviceBridge* class to communicate with the libnifalcon driver: a Java wrapper and a Python wrapper with included examples in lang directory of the libnifalcon library.

A documentation of libnifalcon is generated by a Doxygen documentation system but contains also an introduction, design overview and other useful information which helps a developer. The library contains many examples and even a template of udev rules file that enables libnifalcon communication for non-root users in Linux operating system.

There are a few bugs when using libnifalcon especially on Windows. Firmware loading of a not previously homed Falcon device using the ftd2xx library needs to be repeated a few times in order to let libnifalon believe that the firmware is really loaded or the firmware loading verification is completely omitted. The *getDeviceCount* method returns -1 instead of 0. Comments on bugs can be even found in the source code. The CMake[18] build system does not generate all proper project files for Microsoft Visual Studio and manual editing of the project is needed.

Libnifalcon provides a good alternative to the original driver and SDK. The driver may not be as fast and efficient as the original one but offers cross platform support. Libnifalcon is mostly integrated as a low-layer interface for Linux and Mac OS X operating systems in high level APIs such as H3D API. Integration of the libnifalcon driver to the CHAI 3D is a part of this thesis described in chapter 4.

3.6 HAPI rendering engine

HAPI is a new complex open source low-level haptic API developed by SenseGraphics [19] licensed under GNU GPL v2. Closed source license for commercial use is also available. HAPI is written in the C++ programming language and works on all major operating systems: Microsoft Windows, Linux and Mac OS.

HAPI is one of the most active haptic APIs supporting devices from Sensable, Force Dimension, Novint and Moog FCS Robotics. There are four haptic rendering algorithms available:

[18]`http://www.cmake.org/`
[19]`http://www.sensegraphics.com`

- God-object algorithm - based on the article from Zilles and Salisbury [1] described in section 3.1.2.
- Ruspini algorithm - based on the article from Ruspini et al. [2]
- CHAI 3D rendering - the CHAI 3D API rendering algorithm layer
- OpenHaptics rendering - an OpenHaptics API rendering algorithm layer

HAPI provides not only the basic device handling, but there is also a number of haptic force effects (HapticForceField, HapticPositionFunctionEffect, HapticShapeConstraint, HapticSpring, HapticTimeFunctionEffect, HapticViscosity), surface effects (FrictionSurface, DepthMapSurface, HapticTexturesSurface, OpenHapticsSurface), collision detection (axis-aligned and oriented bounding box trees), primitive shape creation and thread handling.

A very specific functionality is graphics rendering based shape creation. It allows a programmer to create haptic shapes using standard OpenGL drawing functions. A *FeedbackBufferCollector* class collects all triangles that are rendered via the OpenGL library.

HAPI is very well documented with an accompanying manual, reference manual generated by Doxygen documentation system and a lot of examples of all features. The source code of the basic device handling application written in HAPI using the *AnyHapticsDevice* class has just about 20 lines. HAPI can be downloaded as a Windows Installer or as the source code.

HAPI is one of the commercially developed open source API with a very good support from authors. The manual and examples make HAPI very easy to use. The HAPI source code and project hierarchy is not as transparent as in the CHAI 3D API, but there's almost no reason to read it at all. HAPI is one of the best choice of commercial and non-commercial low-level APIs.

3.7 H3DAPI scene graph API

H3D API [6] is a high level scene graph API also developed by SenseGraphics. H3D API uses HAPI as a low-level layer for haptics, OpenGL for graphics and the X3D[20] XML-based file format to represent the scene. The

[20] http://www.web3d.org/about/overview/

library is written in the C++ programming language and is licensed under GNU GPL v2. As with the HAPI library, a closed source license is also available.

3.7.1 X3D

The most interesting feature H3D API provides is scene definition in X3D file format. The whole scene with a camera set, lights, primitive objects, complex meshes, textures, etc. is defined as XML nodes. As X3D is originally web-based technology, a texture or any other object loaded from a file can have a URL path.

The haptic device is defined through a *DeviceInfo* node with the haptic renderer specification, position calibration and the proxy model appearance. H3D API implements all HAPI haptic rendering functionality to the X3D specification. For instance, to add a frictional surface effect to the shape in the scene, a XML node *FrictionalSurface* is added to the apparance node of the shape with appropriate properties.

H3D API also supports X3D routes which makes it possible to read data from one source and route it to a specified destination. That is for instance routing the position of the mouse from the *MouseSensor* node to the shape node position. A *PythonScript* node allows to route data from X3D to Python programming language functions.

3.7.2 Python interface

H3D API propose a very unique way of haptic programming using Python scripts on top of the X3D scene definition. A Python interface to the H3D API implements X3D creation and write functions, special bindable node access (haptic device info, viewpoint, etc.) and X3D field types so that it is possible to create a comprehensive application just using the X3D and Python when there's no reason to develop efficient real-time application.

3.7.3 Scene graph and C++

H3D API is not only the Python and X3D. The entire application can be written in the C++ programming language for better performance. The C++ code allows to parse X3D strings which makes it easier to create objects or set materials in C++. This method should be used only in initialization of the scene because real-time X3D parsing in a graphics loop of the application would lower the performance.

H3D API is a perfect tool to create fast prototypes of applications using haptics. Python and X3D is available for a very rapid development and C++ for higher performance applications. H3D API has much in common with HAPI, it contains a good manual [7], reference guide and examples. The API can be downloaded as a Windows Installer or as a source code package.

3.8 OpenHaptics toolkit

OpenHaptics [8] is a commercial software development toolkit specifically designed for SensAble devices written in the C++ programming language. The toolkit is available only for people or organizations that have bought the device from SensAble with appropriate license. Academics Edition for eligible educational institutions can be downloaded for no charge.

The OpenHaptics toolkit is divided into these layers:

- QuickHaptics micro API
- Haptic Library API (HLAPI)
- Haptic Device API (HDAPI)
- PHANTOM Device Driver (PDD)

QuickHaptics micro API offers a quick development of haptic applications using a high level scene graph. A shape with many properties (texture, draggable option, spinning, translation, ...) can be added to a *DisplayObject* class that handles a display window. QuickHaptics also provide deformable object support and dynamics simulation. The shape can be set deformable just by calling the *dynamic* method of the *TriMesh* class. Gravity of the shape is turned on or off by calling the *setGravity* method.

HLAPI is a high-level API with the main aim of easier integration of haptics into existing graphics application. It provides mapping of haptic workspace, shape rendering or surface and force effects. A feedback buffer of OpenGL can be used to capture graphics primitives, as presented in HAPI. A simple callback system of touch events in the scene is also implemented.

HDAPI is a low-level API that handles supported SensAble devices. As every low-level API, it manages the initialization of the device, servo loop, position, rotation and force update. There are two types of callbacks siilar to the HDAL API:

- synchronous call - a blocking call which returns the current state of the device
- asynchronous call - a non-blocking call which is often used in a haptic loop to get the latest state of the device

OpenHaptics is a very comprehensive toolkit for SensAble devices. A complete manual which familiarizes a developer with all layers called OpenHaptics Toolkit Programmer's Guide is a very good resource for learning OpenHaptics.

3.9 Summary

A table of API specification presents a summary of haptic APIs with a grading from 1 (the best) to 5 (the worst). Development state is an overall state of the project regarding updates, activity and fixed bugs. Devices support grade specifies how many devices from different manufacturers does the API support.

API	CHAI 3D	HDAL	JTouchToolkit	libnifalcon	HAPI	H3D API	OpenHaptics
Open source	Yes	No	Yes	Yes	Yes	Yes	No
Cross platform	Yes	No	Partially	Yes	Yes	Yes	Yes
License	GPL v2/Com.	Com./Non-com.	GPL v2	BSD	GPL v2/Com.	GPL v2/Com.	Com./Acad.
Development state	2	1	4	2	1	1	1
API manual	No	Yes	No	No	Yes	Yes	Yes
API reference	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Examples	Yes	Yes	No	Yes	Yes	Yes	Yes
Devices support	2	4	3	4	1	1	3
Overall grade	2	3	4	2	1	2	3

Chapter 4

Libnifalcon implementation in CHAI 3D API

The CHAI 3D API supports many different devices. As the thesis is primarily focused on the Novint Falcon device, one of the practical result of the thesis is an implementation of libnifalcon support into CHAI 3D.

The main reason for implementing libnifalcon to CHAI 3D is cross-platform support of the Novint Falcon device in CHAI 3D and a possibility to make CHAI 3D free of proprietary parts.

4.1 Device support structure in CHAI 3D

The CHAI 3D API is a modular API with layered architecture. A *cGenericDevice* class is an abstract class for device communication. This class is inherited by *cGenericHapticDevice* which is an abstract class of haptic device used by the CHAI 3D haptic tool class *cGeneric3dofPointer*. A *cLibnifalconDevice* class implements the libnifalcon library support into the CHAI 3D device architecture as shown in figure 4.1.

4.2 Implementation

A preprocessor definition of the USB communication library was created, because there are two different choices of libraries. When the symbol

```
#define ENABLE_LIBNIFALCON_FTD2XX_SUPPORT
```

is defined, the FT2XX library is used, otherwise (which is a default behavior) libusb 1.0 is used.

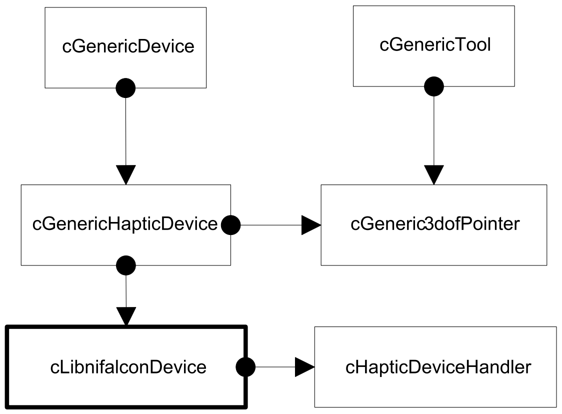


Figure 4.1: CHAI 3D device use architecture

A constructor of the class decides whether the system is available by determining a number of working devices supported by libnifalcon. The main problem in the libnifalcon library (described in section 3.5) is the firmware loading on Microsoft Windows operating system. There's a loop which tries to load a firmware five times and continues even if the firmware does not load properly. The process of haptic device initialization is printed to the standard output.

The class is forcing the homing mode of the device in every call that runs the servo loop. The LED diagnostics of homing provided by the Novint driver do not work in the libnifalcon driver which may be confusing. Switches on the grip of the Falcon devices are assigned in the same way as the original Novint driver does.

4.3 Benchmark

A special purpose haptic benchmark utility was created in CHAI 3D to test libnifalcon performance. Implementation details of the benchmark utility are given in section 5.5.

The benchmark consists of three tests:

- Test 1 - single haptic loop frequency
- Test 2 - multiple haptic loop frequency
- Test 3 - position resolution

In the first test, there's only one thread that reads the current position of the haptic device and measures the frequency of the haptic loop. The frequency reflects a performance overhead of the communication, kinematics algorithm and system calls.

The second test starts a thread where all currently available haptic devices are used. This test shows a practical application where more than one device is used in the haptic loop.

The third test measures attainable position resolution of the device on the basis of haptic tool movement. A minimal, average and maximal distance between positions of two successive servo loops is calculated.

4.4 Result

The benchmark was tested on a dual core Intel Atom processor clocked at 1.6 GHz with Hyper-threading support, 2 GBs of RAM running Microsoft Windows 7 32-bit operating system and Ubuntu Linux 9.10 Karmic Koala. The first haptic device was a Novint Falcon device, the second device was a CHAI 3D debug device (*cDebugDevice*) which is the virtual device with extra *setPosition* and *getForce* methods created for the purpose of CHAI 3D high level testing application described in section 5.2.

On the Microsoft Windows operating system the original Novint driver achieved approximately 3546 KHz with the Novint Falcon device and 16 MHz with the debug device in the first test. The second test lowered the performance of both devices by 10%. A debug build of the benchmark utility lowered the performance by 70%.

The libnifalcon driver using the FT2DXX library on the same system had a performance of about 90% of the original driver and the position resolution test was almost identical.

The benchmark provided interesting results on the Ubuntu Linux operating system using the libnifalcon with libusb 1.0 library. The first haptic loop test achieved a frequency of exactly 1 KHz, which is the frequency of the Novint Falcon device. The position resolution test showed that the Linux implementation of libnifalcon may not be as accurate as the original Novint driver. However, this test is not exactly precise, because the user may move the grip of the device at a different speed which causes a difference in position resolution calculation.

Chapter 5

Haptic API suite

Haptic API suite is a collection of testing applications created to show a basic functionality of haptic APIs presented in this thesis that support the Novint Falcon device.

5.1 Content of the suite

The Haptic API suite tries to cover most APIs usable with the Novint Falcon device. Low-level APIs such as HDAL SDK and libnifalcon support just the Novint Falcon device. High-level APIs support various devices from Sensable, Force Dimension or HapticMaster.

Figure 5.1 shows Haptic API suite system architecture layers starting from the Novint Falcon device at the top and ending with the scene graph high-level APIs at the bottom. White boxes represent APIs, light grey boxes are classes wrapping the driver or low-level API and dark grey boxes represent abstract device classes within the API.

Next, Figure 5.2 represents all testing applications in the Haptic API suite. As in the previous figure, white boxes represent haptic APIs. Testing applications are in grey boxes connected with the API used.

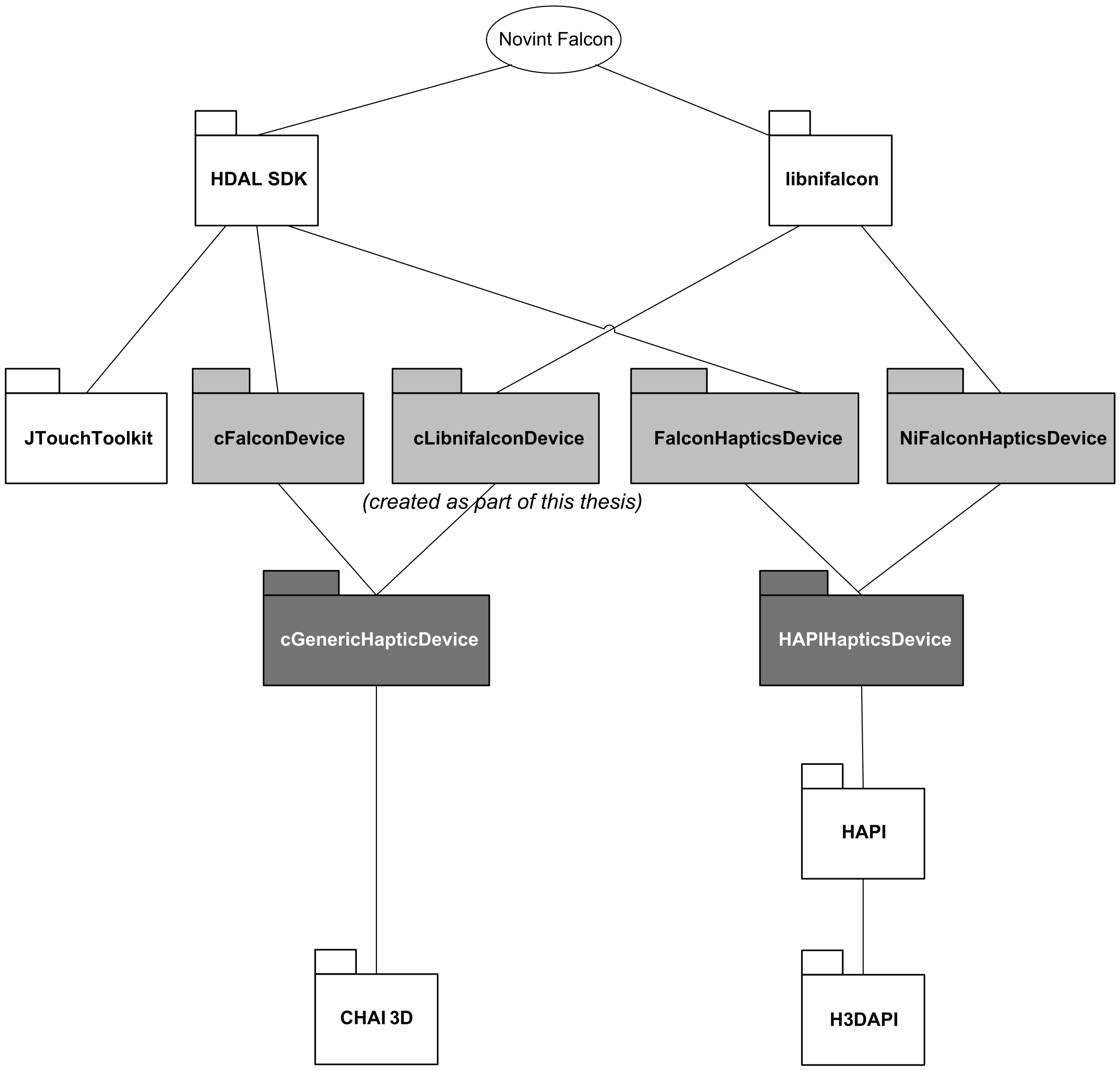


Figure 5.1: Haptic API layer schema

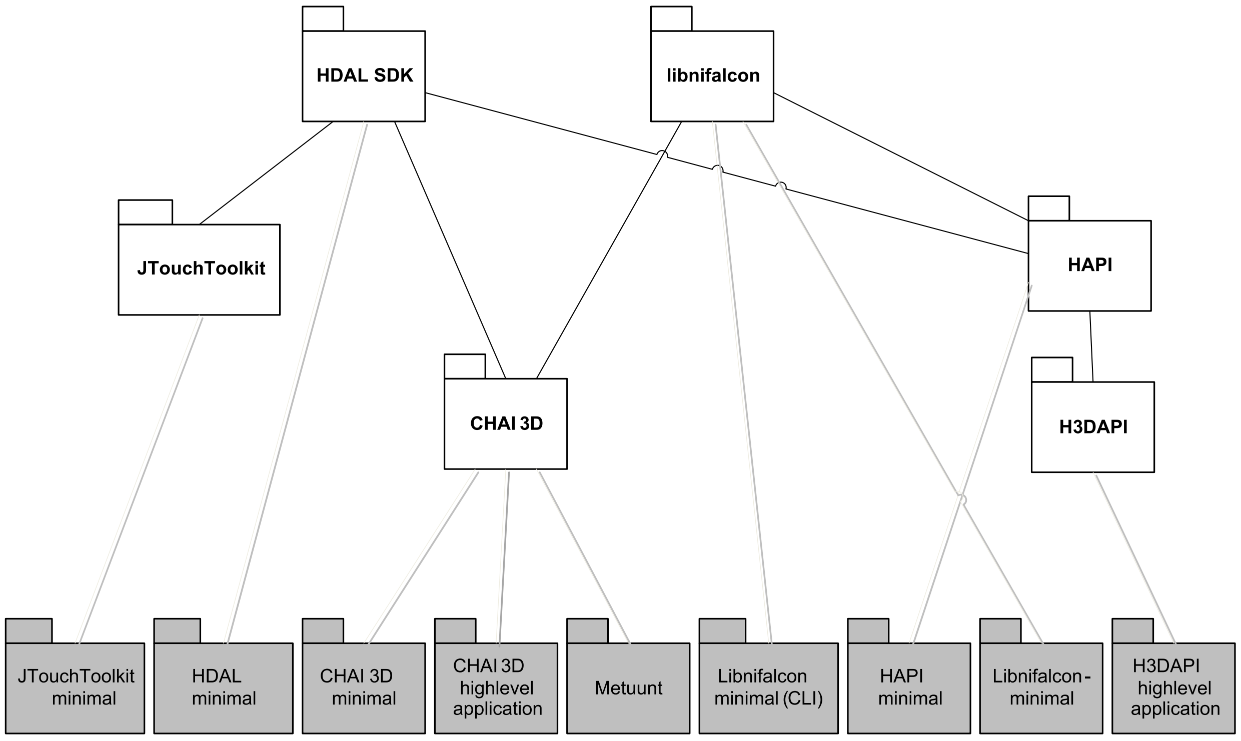


Figure 5.2: Content of the Haptic API suite

5.2 CHAI 3D - highlevel application

The aim of the CHAI 3D - highlevel application is to show possibilities of the CHAI 3D scene graph API in more detail. The main purpose of the application is to use most of the functionality that CHAI 3D offers to the developer. To satisfy this requirement, the application provides a graphical user interface for general scene graph methods and can be described as a 3D haptic prototyping application. The source code follows some of the code convention and aspects of the CHAI 3D API.

5.2.1 GLUT library

CHAI 3D provides OpenGL graphics rendering but it won't handle opening windows, nor process input devices (mouse, keyboard, etc.). The CHAI 3D high-level application uses one of the simplest libraries called OpenGL Utility Toolkit (GLUT). GLUT manages the cross-platform window creation, input device processing and graphics loop. GLUT is a C library and all updates (render, input) are managed via C function pointer callbacks that have to be global or static. The file system.cpp contains all system calls to and from GLUT such as *updateGraphics*, *updateMouse* and even an *UpdateHaptics* function.

5.2.2 Threads

The CHAI 3D high-level application uses a main thread (graphics rendering, application logic computing) and a special haptic thread. The main reason for creating an extra thread is the sample rate of the haptic device. Novint Falcon programmer's guide states that the application has approximately a millisecond to calculate the forces to achieve a credible haptic effect. The time spent on haptic calculations is called a haptic step in the CHAI 3D high-level application and is shown in the status bar at the top of the screen.

A haptic tool uses and modifies resources that the main thread is working with. Adding a new mesh into the scene within a haptic thread without locking the main thread can cause miscalculations and the application might start to behave unstable or even throw a run-time error. The Boost *scoped_lock* class instance locks the variable *m_lock* for the whole scope of a code for all threads. If any other thread tries to lock the*m_lock* for itself it has to wait for the other thread to unlock this variable. The simplest spin-lock algorithm is sufficient enough because there are just two threads in the application and the operation takes reasonable time to complete.

5.2.3 cScene and cConfig classes

The CHAI 3D high-level application is divided into five main classes. The most important class *cScene* defines the whole scene, i.e. the world of objects, physics world, cameras, movement, haptics. On one hand it is good to split functionality into several classes, but on the other hand there's often a need of communication between them. This very complex communication dependency makes the source code unmanageable. One solution is to create a singleton class that will create a communication interface. This solution has its negative impact on the code consistency because it is possible to access the scene from everywhere. It is then up to the programmer to use the singleton interface carefully and only when really needed. Another singleton is a *cConfig* class. It provides access to the configuration such as window width, height or current mouse position.

5.2.4 cHaptic class

A *cHaptic* class provides methods to initialize haptic devices, update haptic devices data and haptic tools as well as methods to obtain specific information about the haptic device. One of the aspects of the CHAI 3D high-level application is that it is possible to use more haptic tools at once. For this reason there's a standard vector class of tools included in this class.

The *cHaptic* class also manages a 2D user interface cursors to control a hud menu at the bottom. Part of the project was also integration of a new virtual debugging haptic device into CHAI 3D because the original virtual haptic device works only on Microsoft Windows operating systems. This device class is called *cDebugDevice* and as the generic haptic device provides methods to get position and set force, the *cDebugDevice* provides *setPosition* and *getForce* methods to control the haptic device. The update method of the *cHaptic* class contains code to update the *cDebugDevice* with a mouse by casting a generic device to *cDebugDevice* class.

5.2.5 cHud class

The fourth class is the *cHud* class that provides the interactive menu interface at the bottom of the screen. Every state of the menu is defined by a *sHud* structure that contains the name of the menu, menu items and a CHAI 3D generic object. Clicking the hud invokes the *onClick* event and the *processHudAction* method is called with the arguments of hud type, action (specified by the clicked item or empty if no item was clicked), item index and haptic index which is fundamental to the interface itself because some actions (e.g. fly-by spectator camera) need to know which haptic device initiated the action.

5.2.6 cCameraSet class

The *cCameraSet* abstract class defines the interface for camera interaction. Every camera type has to inherit the *cCameraSet* class to work properly. The update method takes only one attribute - *stepTime* - which defines the time between the last frame rendered and current time.

There are five types of cameras in the application:

- Default camera - a camera set on the start of the application
- Static camera - a static camera that remembers the last direction
- Observer camera - a camera that observes a haptic tool
- Object camera - a camera that observes a selected object
- Spectator camera - a fly-by spectator camera

5.3 Metuunt project - CHAI 3D - lowlevel application

The project Metuunt intended to be a basic 3D engine for MMORTS game. It was a semester project for a programming course.

Project Metuunt was used for a testing application that would implement haptic support into an existing project that had no plans to include such a support at a time of development.

Metuunt is divided into several classes providing special functionality, i.e. loading 3DS model file format (C3ds.h), octree frustum culling [9] (COctree.h), input device processing (CInput.h), high-precision timer support (CTimer.h), configuration (CConfig.h), logging mechanism (CLog.h), world object management (CObject.h), graphics management (CGraphics.h) and the scene containing objects (CScene.h).

The global instance of *CSystem* class System provides communication interface. The application uses WINAPI functions to manage window creation and DirectX 9.0c for rendering the scene. The important file for the Haptic API suite is CHaptic.h containing *CChai* class definition. It manages the initialization, receiving haptic position for the spectator camera and calculation/normalization of global forces.

The basic object sphere collision (this can be tested against the church building) is defined in the *CObject* class itself. Providing the three-dimensional vector of a collision tool the *CObject::getCollisionForce* method returns the normalized force. All forces are summed up and sent to the device by invoking the *CChai::SetForce* method. If the size of the force is greater than the maximum force allowed for the specified haptic device the update algorithm simply normalizes this force.

5.4 CHAI 3D - minimal application

CHAI 3D - minimal application is an application that shows the very basic use of the API.

As this project aims to be really minimalistic, it does not used any separate haptic thread (which CHAI 3D supports too) and it simply runs an infinite loop which ends on a haptic button press. CHAI 3D provides classes to work with three-dimensional vectors *cVector3d*. In order to get the position of the haptic device it simply calls a method *cGenericHapticDevice::getPosition* and passes the reference to the position vector as a function parameter. The last segment of code detects user switches (haptic button press) and checks whether the position has not changed so that it doesn't have to print the same coordinates again.

This example doesn't show the way how to send forces to devices. This is simply done by calling the method *setForce* within the cGenericHapticDevice instance with appropriate force vector.

5.5 CHAI 3D - haptic benchmark

CHAI 3D - haptic benchmark is an application developed for libnifalcon library performance testing. The application uses the same functionality of CHAI 3D as CHAI 3D - minimal application and the Boost library is used for thread handling.

As described in the section 4.3, there are three tests in the benchmark. The first test starts an extra thread for all available devices and measures the frequency of haptic loop for five seconds.

The second test creates a thread for every device. Every thread waits for a synchronization flag called *threadHandler1* and then runs the haptic loop with frequency measuring.

The third test measurers the attainable position resolution for every device by computing the distance that the haptic tool covered in the workspace in each iteration of the haptic loop. The user has to move the haptic tool adequately to get correct results.

5.6 HDAL - minimal application

Project HDAL - minimal is an application that uses the HDAL proprietary driver and SDK. It does not treat the situation of a missing Novint Falcon driver in the system and the application may throw an access violation error. The purpose of this application is to show the way of accessing and sending data to the haptic device using the HDAL SDK.

After the successful initialization, the access to the device data can be obtained in two separate callbacks as described in the section 3.3.

5.7 JTouchToolkit - minimal application

JTouchToolkit API does seem to be a dead project but it still provides quite a good way to develop Java application that use a haptic device.

Java platform may be operating system independent but the support of the Novint Falcon is strictly Microsoft Windows dependent because it wraps the Novint HDAL driver. To use a Novint Falcon on the Linux operating system, please see the libnifalcon library Java wrapper.

It is possible to compile JTouchToolkit - minimal in Microsoft Visual Studio with a custom tool using javac.exe that has to be set in the PATH environment variable. Please use a native Java IDE for better debugging and other tools. JTouchToolkit contains libraries JHDAL.dll, JHDAPI.dll and JHLAPI.dll that need to be in the working directory to run the application.

JTouchToolkit uses a method of adding haptic listeners very similar to HDAL callbacks without a need of setting blocking or non-blocking access. A class that can operate with the device is created by implementing *HapticListener* class and overriding appropriate methods. Initialization of the device is done by a static function *FalconDevice.newFalconDevice* with a function parameter specifying the index of the device. The number of devices is returned by a static function *FalconDevice.countDevices.*

5.8 libnifalcon - minimal applications

The libnifalcon library provides a very fast way to test haptic applications. This part of the API is the framework *FalconCLIBase* that manages all the initialization, command line parsing and firmware loading on its own.

To use *FalconCLIBase*, the programmer just needs to inherit the class and override methods *addOptions* and *parseOptions*. The *runLoop* method is the standard haptic loop method for getting a device data and setting the type of libnifalcon kinematics. A few bugs can be found in the *FalconCLIBase*, e.g. *help* program option always prints the name of the application as falcon_test_cli instead of a real application name.

The minimal application not using the *FalconCLIBase* framework is also available.

5.9 HAPI - minimal application

The HAPI - minimal application is the smallest low-level haptic API application (in source code length). The HAPI library contains an *AnyHapticsDevice* class that initializes any available haptic device by calling *initDevice* and *enableDevice* methods. The *HAPI::HAPIHapticsDevice::DeviceValues* structure is then used to work with the haptic data.

5.10 H3DAPI - highlevel application

The H3DAPI - highlevel application shows the basic use of the X3D and Python interface of H3D API. The application uses an *AnyDevice* X3D node to work with the haptic device and the God-object algorithm to render haptic shapes. The scene contains one animated sphere with a frictional surface effect and one mouse proxy sphere. The haptic tool defined as *stylus* is also represented as a sphere.

Chapter 6

Conclusion

6.1 Summary

The thesis has introduced the haptic technology as human-computer interaction along with possible practical applications and the Novint Falcon device was presented as the main haptic device used in the survey.

Diversity of Haptic APIs architecture was analyzed and the survey described three different abstraction layers of haptics programming. The CHAI 3D library was examined in more detail with a description of the scene graph, haptic tool, force effects, rigid body dynamics simulation module and deformable body simulation module. A technology, licensing, documentation, development state and a functionality specification of all haptic APIs was presented and summarized in the section 3.9.

The libnifalcon library was successfully implemented to the CHAI 3D API with some minor problems and the benchmark application developed for performance testing provided data to compare the device manufacturer driver and libnifalcon driver.

Testing applications that demonstrate basic use of APIs together with a larger application presenting the CHAI 3D library with some added features were created and included in the Haptic API suite.

6.2 Possible future extensions

The Practical Survey of Haptic APIs was primarily limited by the number of haptic devices. The only haptic device available at the faculty was the

3-DOF Novint Falcon device. A PHANTOM Desktop 6-DOF haptic device from SensAble will be available during the 2011/2012 academic year which would allow to analyze more APIs, especially the complex OpenHaptics toolkit.

The very promising H3D API and HAPI libraries could also be examined in more detail.

Appendix A

Contents of the accompanying CD

The structure of accompanying CD is as follows:

- /bin - contains compiled binaries of demonstrated programs for Microsft Windows XP (or newer) operating system along with application data files
- /doc - contains the bachelor thesis in PDF format and the Haptic API suite documentation
- /ext - contains external libraries used in the Haptic API suite
- /src - contains source code of the Haptic API suite together with a Microsoft Visual Studio 2008 solution and project files
- install.exe - is an installer of the Haptic API suite

Bibliography

[1] Zilles C. B., Salisbury J. K.: *A Constraint-based God-object Method For Haptic Display*, ASME Haptic Interfaces for Virtual Environment and Teleoperator System, 1994.

[2] Ruspini C. D., Kolarov K., Khatib O.: *The Haptic Display of Complex Graphical Environments*, In SIGGRAPH 97 conference proceedings, volume 1, pp. 295-301, August 1997.

[3] CHAI 3D library `http://www.chai3d.org`

[4] Novint Technologies Incorporated *Haptic Device Abstraction Layer (HDAL) Programmer's Guide*, 2008.

[5] libnifalcon library `http://libnifalcon.sourceforge.net`

[6] H3D & HAPI libraries `http://www.h3dapi.org`

[7] SenseGraphics AB: *H3D API MANUAL*, 2009.

[8] Openhaptics toolkit `http://www.sensable.com`

[9] Foley D. J., van Dam A., Feiner S. K., Hughes J. F.: *Computer Graphics: Principles and Practice in C (2nd Edition)*, Addison-Wesley, Proffesional, 1997.