Force-Feedback Hand Controllers for Musical Interaction

Stephen Sinclair

Music Technology Area
Schulich School of Music
McGill University
Montreal, Canada

June 18, 2007

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Arts.

2007/08/09

Abstract

This thesis investigates the possibility of exploiting haptic force-feedback technology for interacting with virtual musical instruments. A survey of current software solutions for creating haptic virtual environments is provided, with a discussion on the need to integrate such a platform with currently accepted solutions for audio research.

A system was developed to combine a haptic programming library with a physical dynamics engine and to expose its functionality through the Open Sound Control (OSC) protocol, an increasingly accepted standard for communication within the audio software and hardware domain. Using OSC messaging, simple 3D objects can be instantiated and constraints on their movement can be specified, allowing the description of physically dynamic mechanisms. Collision events as well as properties of the objects can be transmitted to the audio system continually to be used for modulating audio synthesis parameters. Some examples of simple virtual musical instruments created with the aid of this system are provided.

Resumé

Ce rapport de thèse examine la possibilité d'utiliser la technologie haptique basée sur le retour d'effort dans l'intéraction avec des instruments de musique virtuels. La revue des logiciels courants pour la création d'environnements de réalité virtuelle incluant des sensations haptiques a montré un manque de prise en compte fonctionnalités audio. Il est discuté le besoin de faire un lien avec les logiciels utilisés couramment en recherche acoustique.

Un système a été créé pour combiner les fonctions d'un logiciel d'haptique et d'un moteur physique, en exposant leurs fonctionnalités à l'aide du protocole Open Sound Control (OSC), un standard de plus en plus utilisé pour la communication entres systèmes d'audio logiciels ainsi que matériels. Des objects simples en trois dimensions peuvent ansi être créés et soumis à des contraintes de déplacement, permettant la description de mécanismes physiques. Les propriétés de ces objects peuvent êtres manipulés sur une base d'évènements ou d'actions continuelles, et être utilisés pour controller les paramètres de synthèse sonore. Quelques exemples d'instruments musicaux virtuels sont implémentés à l'aide de ce système à titre d'illustration.

Acknowledgments

This work could not have been completed without the help of many wonderful individuals who surround me. Whether their role has been to actively discuss ideas, to create distractions that enabled me to think laterally, or just to lend some moral support, my thanks go out to those who have been there to help this work along.

First and foremost I want to thank my father, Ian Sinclair, who is responsible for introducing me to computing at such a young age and eventually to robotics and haptic technology through working together at MPB Technologies, and my mother, Meg Sinclair, who put up with me spending so many summers indoors hacking on code and making weird electronic music instead of playing outside like a normal boy.

I'd like to thank all those who have made my academic experience at McGill a pleasant and inspirational one. My supervisor, Marcelo Wanderley, is a bottomless cup of knowledge on the subject of musical gesture, and his always-positive demeanor and limitless suggestions for how to improve my work have proven incredibly useful and encouraging. His uncanny ability to handle (and shield his students from) administrative tasks has also been more than appreciated. All the other professors in the music technology area also deserve some thanks for being cheerfully helpful whenever I had a question or needed a favour.

The students in the IDMIL and the rest of music technology have been wonderful to share lab space with, proving that friendship and work aren't at all mutually exclusive. In particular I'd like to thank Joe Malloch for our unending (but fruitful) discussions on how best to deal with Open Sound Control. Thanks also go to the mass of open-source developers who have provided me with so many amazing tools that have allowed me to complete my work quickly and efficiently.

I'd also like to thank all my friends outside of school who forced me to once in a while stop doing homework and to get out and relax. Last but not at all least, I want to thank Shabana for her moral support. She has repeatedly reminded me that I was doing the right thing to undertake a graduate degree, and I am inspired by her intelligence, beauty, and creativity every day. I only hope that she and everyone I've mentioned is ready to deal with me for three more years of student life while I pursue my next academic goal.

This work was sponsored in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canadian Foundation for Innovation (CFI), and the Enactive Network European Project.

Contents

List of Figures

List of Tables

List of Acronyms

ACROE	*Association pour la Création et la Recherche sur les Outils d'Expression*
API	Application Programming Interface
CAD	Computer-Aided Design
CHAI	Computer Haptics & Active Interfaces
COM	Component Object Model
CPU	Central Processing Unit
DIMPLE	Dynamically Interactive Musically PhysicaL Environment
DOF	Degrees of Freedom
DSP	Digital Signal Processing / Digital Signal Processor
FM	Frequency Modulation
FSR	Force-Sensing Resistor
GEM	Graphics Environment for Multimedia
GPL	GNU General Public License
HCI	Human-Computer Interaction
JND	Just-Noticeable Difference
LGPL	Lesser GNU General Public Licence
MIDI	Musical Instrument Digital Interface
ODE	Open Dynamics Engine
OSC	Open Sound Control
OSG	Open Scene Graph
TCP/IP	Transport Control Protocol, Internet Protocol
TGR	*Transducteurs Gestuels Rétroactifs* (Retroactive Gestural Transducers)
UDP/IP	User Datagram Protocol, Internet Protocol
URN	Universal Resource Name
VMI	Virtual Musical Instrument
VRML	Virtual Reality Modeling Language
VR	Virtual Reality
WIMP	Windows, Icons, Menus, Pointer

Chapter 1

Introduction

1.1 The Haptic Sense

It has been known for some time that the *feel* of a instrument plays an important role in a musician's ability to learn and to play it [71]. Indeed, our ability to feel the surrounding environment, our *tactual*, or *haptic* sense, is used constantly in our daily interactions with the world. The sense of touch can be roughly divided into two general categories: the tactile or *cutaneous* sense, which allows us to feel *textures*, *vibrations*, and *temperature differentials*; and the kinesthetic sense, also called *force feedback*, which allows us to internally feel the positions of our limbs through feedback from our muscles, as well as to externally feel the resistance that an object exerts on our muscles [10].

The *synthesis* of haptic feedback in the digital domain, what Gillespie [33] described as "computer mediated emulation of mechanical impedance," is of growing interest to researchers in virtual reality. Visual and auditory feedback have seen much attention in the last few decades; despite having had some attention as early as the late 1970's [13], comparatively speaking, haptics is still a young field. This is largely due to, though certainly not restricted to, the mechanical and computational demands of haptic display. However, the results are worthwhile: the combination of haptics with audio and visual feedback, a so-called *multi-modal* display, provides a far more immersive virtual experience for a user than a display providing only one or two sensory modes [10]. Additionally, the *haptic channel* can provide a means of communicating state information to the user of a device without relying on or interrupting visual or auditory streams [39].

1.2 Gestural Controllers and Virtual Musical Instruments

Ever since the introduction of electronics into the musical domain, musicians and researchers have struggled to find convenient ways to provide control for electronics that are more intuitive than the traditional knobs, sliders, mice and keyboards that have become ubiquitous in the digital era. It has been recognized that the instrumental gesture is key to the expressive production of sound. Cadoz [11] claimed that the instrumental gesture is inseparable from the sound, and that it in fact *defines* the sound.

Correspondingly, an impressive number of controllers have been built in the last few decades that all make an attempt to extract gestural information for use in sound synthesis. Mulder [63] summarized these into categories such as "touch controllers", "expanded range controllers", and "immersive controllers". Other terms have been used such as "hands-free" [62] or "open-air" [80] controllers. All of these use some form of sensing to transduce human movement into digital signals which can control sound synthesis algorithms. Wanderley and Depalle [94] considered that the combination of a gestural controller and a sound synthesis, two processes that traditionally have an inherent relationship for acoustic instruments, can be thought of as a single unit termed a *digital musical instrument* (DMI).

While gestural controllers have been able to help restore a relationship between instrumental gesture and sound synthesis, this relationship is arbitrated by a mapping layer between controller output parameters and synthesis inputs. Development of this mapping layer is not necessarily trivial, and is an important part of the DMI [42]. The task of creating and perfecting it can be a long process consisting of constant feedback between performer, instrument, and designer [53].

Citing the difficulty of adapting physically-based gestural controllers to the needs of different performers, Mulder [61] proposed that the DMI should be abstracted by one more level. He suggested that a *virtual* musical instrument (VMI)—a gestural controller defined in software—would be the most flexible controller possible, and would allow gestural access to "all possible audible sounds". While somewhat understating the difficulties of implementing a generalized software-based controller paradigm that would allow unlimited access to an arbitrary parameter space, the idea that Mulder was proposing was quite sound: the introduction of a new mapping layer from the physical controller to a virtual controller, which would in turn have outputs mapped to a sound synthesis engine. In particular for "hands free" controllers, like the datagloves used by Mulder for the control of

Fig. 1.1 Axel Mulder demonstrating glove-based interaction with a virtual musical instrument, developed with Sidney Fels and Kenji Mase, at *Kansei—The Technology of Emotion Workshop* in 1997, Genova, Italy [64], from *Trends in Gestural Control of Music* [63]. A close-up of the "rubber sheet" object is seen on the right-hand side, from [61]. Both images are reproduced here with permission.

his VMI, the extra abstraction layer enabled a performer to use metaphor to help visualize the control surface of an instrument that he could not actually touch. (See Figure 1.1 for a picture of Mulder playing a virtual instruments.) This created a meaningful and concrete mapping between hands-free gestures in three dimensions to a virtual space which could allow fewer or more degrees of freedom than the physical controller could actually provide. In other words, the VMI helped to provide a semantic grounding to an otherwise arbitrary many-to-many relationship between controller and sound.

1.3 Force-Feedback Displays as the VMI Physical Layer

I will now review some applications of haptic display, and discuss how similar ideas might be applied to musical purposes. Salisbury [82] proposed several uses for force-feedback haptics: seismic modeling, virtual prototyping, shape sculpting (for 3D modeling), molecular docking, and surgical simulation and training. Many of these ends have since been pursued. To date, force-feedback has found its way into gaming devices, rehabilitation for the visually impaired, desktop user interfaces, visual art (painting and 3D modeling,) surgical simulation and telesurgery, and virtual prototyping of CAD designs [46]. It has also been

used in music research [13, 29, 66, 72, 90], which will be discussed in depth in the next chapter.

Generally speaking, force-feedback haptics sees many uses in areas involving virtual reality. This is because force feedback requires direct user interaction for display. In order to "display" something, the end effector of a haptic device must be moved by the user so that it can be repelled. As the user explores by probing with the end effector, consistancy in the "force field" can give an idea of a solid object. The implication is that in order to provide force feedback, some idea of *what* is being displayed is usually modeled by the computer—a virtual environment.

Thinking back to gestural interfaces, a controller is simply an object to be manipulated by a user, so that instrumental gestures can be transformed into sound. As Mulder showed, the controller itself can be virtualized in a similar way that a CAD design can be manipulated on the computer before being built, or your left knee can be simulated for surgical training. With force feedback getting so much attention in these applications, it seems natural to think about how it could be similarly applied to create a more immersive VMI experience.

Mulder stated repeatedly in his dissertation that his VMI unfortunately lacked tactual feedback [62]. He used a pair of input devices called datagloves which could sense the position of each hand in space as well as the bend of each finger. Using deformable physical models computed in Max/FTS, users could use their hands to push and pull on the VMI shapes, which changed corresponding sound parameters in real time. At the time, no glove-based haptic devices were available, so datagloves could only provide sensing capabilities. Consequently, users depended entirely on a visual display and the auditory feedback to orient themselves in the virtual environment and accurately perceive the metaphor they were manipulating.

Today, it is possible to purchase datagloves that allow some form of force feedback.[1] However, the problems of actuating the many degrees of freedom in the human hand in a comfortable manner make such devices heavy and prohibitively expensive. Instead, another class of haptic displays, so-called "pen-based displays," have seen a much wider adoption by the commercial market. Pen-based force-feedback hand controllers ranging in capabilities are now developed and sold by at least 5 different manufacturers [28, 32, 60, 67, 84]. Examples of interaction with a pen-based display can be seen in the figures in Chapter 4.

[1] The CyberGrasp from Immersion, for example.

This market competition has reduced the price of force-reflecting 3D manipulators to levels that are becoming more affordable. Thus, although they are mostly limited to interaction with a single point in space, pen-based displays are an excellent starting point for exploiting force-feedback haptics in VMI interaction.

1.4 Motivation and Requirements for a Haptic VMI Implementation

Mulder's main reason for developing the VMI concept was to create infinitely adaptable musical interfaces, but in fact the uses of VMI's, and particularly haptic VMI's, are actually more numerous. From the perspective of musical research, we are interested in exploring what kind of a role the kinesthetic and tactile sense plays in musical interaction. By virtualising the gestural control, it becomes possible to adjust various aspects of haptic feedback in ways that would be impossible with a "real" interface, or even to turn it off completely. With the "feel" of an instrument being variable, we can approach the problem of musical kinesthetics in a properly scientific manner. For instance, haptic parameters such as stiffness, texture, friction, and latency can be modified. A VMI could be played with and without haptic or visual feedback to examine how the presence of a particular sensory mode affects learning and playability. In addition, several haptic "effects" developed in HCI research can be exploited [68]: gravity wells, constrained movement, and vibrotactile cues can be created. Simulations of traditional musical interactions can be achieved, such as bowing, tapping, and plucking. Using physical dynamics, musical instruments that exploit object interaction and movement could be created.

From an industry perspective, there is a need for methods for evaluating and comparing haptic devices. Performance measures for haptic devices are currently known [38], but designing common tests for these measures is an ongoing task. In particular, determining how the needs of haptics for music may differ from, say, surgical applications may help to find VMI that are suitable for testing certain aspects of new haptic devices. Finding expert musicians willing to participate in such experiments should prove more convenient than surgeons, making comparison testing easier.

With these purposes in mind, it can be specified what sort of features to look for in a good VMI implementation. Firstly, since several audio environments are available for real-time sound synthesis, easy integration with these programs is desirable. Secondly, for

the purposes of constructing experiments it is important to offer a wide variety of haptic effects with variable parameters. To establish a virtual controller metaphor, these haptic effects must be associated with objects in a virtual environment, which can be manipulated by some input device. Thirdly, the availability of a physical dynamics engine is desirable for creating mechanisms which can react realistically to user input.

1.5 Summary

This chapter has discussed the background, motivation, and requirements for creating a haptically-enabled virtual reality environment which can communicate with an audio system in order to design virtual musical instruments which can be touched and manipulated by a user. The remainder of this thesis will discuss the implementation of such an environment in more details. Chapter 2 discusses previous work in audio-enabled virtual environments with and without haptic feedback, as well as previous research in the use of haptics for studying musical interaction. Chapter 3 contains a survey of software currently available for the development of haptics and virtual environments. Chapter 4 provides details of DIMPLE, a tool for the creation of VMI's controllable from within existing audio software, and describes some examples of instruments that have been created with it to date. Chapter 5 gives some conclusions and ideas for future work with the system.

Chapter 2

Background

This chapter discusses several previous works on force-feedback in musical interaction and on the development of virtual musical instruments. As we will see, a large amount of attention has particularly been given to the simulation of the bowing gesture. The feel of the piano action has also been extensively investigated. In general, significant focus has been put on creating physically accurate simulations of particular real-world instrumental interactions.

2.1 Previous work on force-feedback haptics in gestural control of music

2.1.1 ACROE

As we have discussed, Cadoz [11] proposed that instrumental gestures are accompanied by inherent haptic feedback. When a musician blows a horn, bows, or plucks a string, he feels the tension and release of pressure in the instrument, and feels the vibration of the resonating body in his limbs and torso. This feedback helps him to control the instrument accurately, and consequently its sound. Cadoz, Florens and colleagues at ACROE set out to create a computer-controlled device that could recreate the feel and sound of an instrument. Called a *Transducteur Gestuel Rétroactif*, or Retroactive Gestural Transducer system (TGR), this project involved the design of special motors, dedicated processors, and a modular mechnical system that eventually resulted in the Modular Feedback Keyboard (MFK) [13], now commercialized as the ERGOS device [28]. The modularity of the design

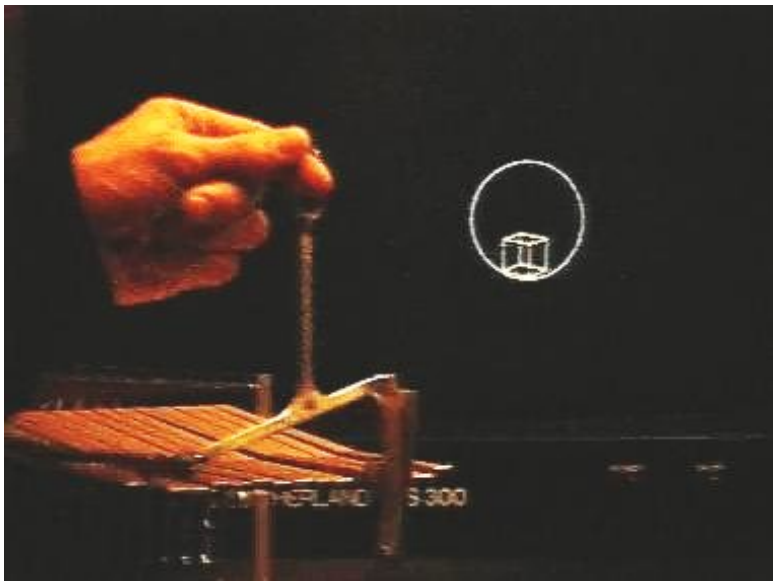


Fig. 2.1 A gestural interaction with the TGR. The user manipulates the circle and can feel the object inside when it collides with the sides. The vibrations of the object give rise to audio feedback. From *Trends in Gestural Control of Music* [12], reproduced here with permission.

allows the mechanical coupling of motors and the interchange of end effectors to create a haptic display with configurable grips and variable degrees of freedom (DOF). In addition to functioning as an actuated keyboard, motions such as bowing and squeezing can be achieved, as well as general 3- or 6-DOF point manipulation. See Figure 2.1 for a picture of the TGR in action.

Some models created with the device include a virtual violin [30], and a child's rattle [31]. These two models were expressed using the CORDIS-ANIMA formalism for physical modeling [14]. This language allows the description of a physical system, which can then be used to interact haptically, generate sound, and be viewed graphically.

In the violin model, the gesture interface, a 3-DOF "horizontal joystick" extended from the MFK keys controls the *exciter* portion of the model. The exciter is connected to a *non-linear links* block which represents the bow-string interaction, and this in turn is connected to a mass-spring model of a string, which can also be replaced by a modal string model. The CORDIS-ANIMA language allows a minimal and exact description of the physical system. It is discussed further in Section 3.1.8. Florens reports that the model has a narrower "space" than a real violin, referring to "the pitch space, the bowing space and the dynamic

of the possible and pertinent bow pressure variations," but that within the playable region the similarity to the real interaction is "striking". By playing with the physical parameters, especially the gesture coupling scale factors, he found that the most comfortable situation corresponds to, "'light' strings that provide a tiny but non-null friction force."

2.1.2 The Touchback Keyboard

Gillespie [33] used a force-feedback system to emulate the action of a grand piano. Commonly, pianists found that the system of weights and dashpots incorporated into electronic pianos were not similar enough to the feel of a traditional piano. While pianists can technically only control the timing and hammer velocity resulting from a key hit, they can produce a range of timbral effects by finely manipulating these parameters. To achieve such control, they place high importance on the "touch" of a piano key, which includes not only the weight, but also a sensation of the underlying mechanism, including the sudden release or "let-off" of the hammer. Gillespie's solution was to simulate the haptic feedback in a piano using digital motor-control algorithms based on a physical model of the piano action. He created the Touchback Keyboard as a result, an 8-key actuated keyboard.[1] His work encompassed a proposed solution to guarantee passivity in the rendering of a virtual wall, a problem which has application across the entire domain of digital haptic display.

Oboe and Poli [69] later created MIKEY, the multi-instrument active keyboard, an attempt to improve Gillespie's simplification of the grand piano action, as well as to emulate the action of a harpsichord and a Hammond organ. They made use of low-cost electronics to show that it is possible to create such an instrument in a cost-effective manner.

2.1.3 Bowing the Moose

O'Modhrain [71] set out to discover whether haptic feedback was critical to musical performance, and if so, whether it is ingrained in a player's internal representation of the instrument. It was observed that a Theremin, an early "hands free" instrument using electric field sensing to allow hand proximity to control pitch, seemed to be easier to play when the player's hand was attached to the antenna with an elastic band. Seeing that the presence of a position-dependent force could help in performance, she proceeded to

[1] It should be noted that he credits Max Mathews as drawing the first sketch of a solenoid-driven actuated key in 1988.

perform several studies on the playability of a Theremin-like sound under various force feedback conditions. The device they used was a 2-DOF force-feedback controller called the Moose, developed by Gillespie and O'Modhrain [34]. It was originally intended as an interface to allow blind persons access to two-dimensional graphical user interfaces. The study tested several linear mappings between pitch and force. Included conditions were: no force, positive and negative constant force, positive and negative spring force, and viscous force. (The last is a change in force related to velocity.)

They found that playing accuracy was better in all force conditions than without force. Additionally, they found, marginally, that the best playing accuracy was in the positive spring condition, where force linearly increased according to pitch. Next they performed another study, this time allowing subjects to practice the melody several times. The intent was to discover whether force only helped in the learning stages of playing an instrument, or if it had a lasting effect. Their results showed that players became accustomed to the force condition, and tended to overestimate their movements if the force was subsequently removed.

The second phase of experiments was to study the bowing gesture. The violin is an instrument shown to provide vibrational feedback through the left hand, which fingers the strings, through the neck and upper body, which are in contact with the instrument's resonating body, and through the right hand, which holds the bow. They implemented a friction model for the tactile response of the bow hand based on research by Hayward and Armstrong [37], excited by lateral movement of the end effector against a virtual wall, which attempted to simulate the stick-slip motion of bow-string interaction. Excitation was accompanied by a physically modeled string sound. In the experiment, subjects were asked to try and reproduce pre-recorded bow strokes. Results were compared analytically as well as by expert judges.

Strangely, they found that novice players performed the same with and without friction, while expert performers performed worse in the presence of friction, despite the fact that all subjects reported preference for the friction condition. The conclusion was that the friction model was not similar enough to the real bow-string interaction, so that it confused players.

Another conclusion one might draw is that perhaps friction plays less of a critical part in the interaction than simply the perpendicular resistance to bow pressure. If the feeling of pressing against a virtual wall was enough to inform players of their instrumental effort, it is possible that lateral vibrations only serve to make the gesture more satisfying, but do

not add any extra information to the interaction. It may be interesting to perform further comparisons under different types of friction conditions. Additionally, the 2-DOF device did not allow investigation of the role of torque, as discussed in Section 2.4.4, which is likely an important component of the bowing gesture.

Relevant to this thesis are some comments in O'Modhrain's conclusion: firstly, she discusses some problems with attempting to control audio and haptic responses with a single process, stating her reasons for choosing parallel processes that communicate information over a MIDI connection. This is relevant to similar design decisions made for our system, discussed in Section 4.4. Secondly, she remarks on the need for a hierarchical control protocol for transmitting gestural information. Again, this relates directly to our decision to use the Open Sound Control protocol for communication with virtual environments. Lastly, she mentions the need for a "development environment for haptic feedback," which would allow for substitution of hardware components and easy access to haptic primitives such as springs, dampers, and friction effects. This has precisely been the goal of the work discussed in Chapter 4, through the use of *objects*, *properties*, and *constraints*.

2.1.4 Nichols' vBow

Inspired by personal musical needs, as well as by the work of O'Modhrain described above, Nichols [66] created the vBow, a motorized bowing machine used to simulate the stick-slip friction of a bowed instrument. The first version of the vBow featured only one lateral degree of freedom. A wire along the "bow", representing the hair, was attached to a single motor and encoder, so that motion along this length could be detected and opposing force could be applied. Subsequent versions supported the addition of rotational, vertical, and longitudinal motion, essentially turning it into a serial robotic arm.

The vBow controls a bowed string physical model designed by Serafin et al. [86]. Citing a large body of work on mathematical modeling of the physical interaction between bow and string, it was made clear that this complex and subtle interface would be difficult to simulate with perfect accuracy. The model used had previously been tested with a Wacom pen tablet input device [85].

An accompanying haptic model is also controlled by the software. Haptic feedback includes "friction and vibration for the lateral motion, detents for the rotational motion, elasticity for the vertical motion, and friction for the longitudinal motion." The vibration

felt through the lateral servomotor is simply a scaled-down copy of the sound synthesis output. On both lateral and longitudinal motions, friction is also felt which is a randomized function that emulates interactions with micro-imperfections in the hair, resulting in a feeling of "non-periodic roughness." The "detent" simulation—the simulation of the bow rolling across the violin strings—is accomplished by providing force feedback in the rotational direction as the cable passes over the location of the simulated strings. A series of "slices", non-linear divisions of the rotational dimension, are used to simulate rolling over each of the 4 strings. Finally, the combined elasticity of the string and bow hair is simulated by a linear spring model in the vertical direction. This model allows the player to feel the tension in the string.

Nichols describes several potential uses for the vBow, including the "bowing" of other types of sound synthesis models, and experimentation with different types of haptic feedback. Simulating different hair and string materials and unusual configurations of strings is suggested.

2.2 Other studies of synthetic tactual feedback for control of sound

2.2.1 Vibrotactile feedback

Chafe [15] showed that introducing *vibrotactile* feedback into a gestural controller can help in maintaining accurate control over the playing of an "unpredictable" physical modelling synthesis. The term vibrotactile feedback can refer to any oscillation felt by the player's body as he plays an instrument. In the gestural controller literature, it is most often used to refer to the playback of the audio signal through a vibrating actuator attached to the body of the controller. In Chafe's case, he attached a voice coil actuator and a strain gauge to a metal bar, which was used to control a wind instrument physical model. The chosen physical model had a "lip tension" parameter, which was mapped to the strain gauge, and consequently to the deformation of the metal. In a traditional wind instrument, the player uses lip tension to control a sound, but also senses the instrument's vibration through his lips. Correspondingly, it was found that when players could feel the sound through the voice coil, they found it easier to play the instrument: specifically, they could more easily determine when the physical model was about to become "unpredictable", and temper their

playing accordingly. Interestingly, Chafe discussed how the pitches of notes were entirely above the frequency range of the human tactile sense. However, on transitions between notes, a brief period of overlap caused subharmonics which could be felt.

Further efforts to explore vibrotactile feedback in gestural controllers include Bongers [9], who described several uses for tactile display in music, and Rovan and Hayward [80], who explored the use of tactile feedback to deliver cues for open-air controllers. The Touch Flute [8]and the Viblotar and Vibloslide [55] are some examples of controllers built at McGill for exploring vibrotactile feedback. Marshall and Wanderley [55] also give a good overview of the effectiveness of various types of actuators for vibration, with reference to the frequency ranges of the human tactile sense.

2.2.2 Force feedback

DiFilippo and Pai [26] created "AHI": an audio-haptic interface using a modified version of the Pantograph device based on a design by Ramstein and Hayward [76] at McGill, and used a dedicated microcontroller to simulate physical collisions with very low latency between audio and haptic responses. The device was a 3-DOF input device, with two linear directions and one rotational direction. (The original Pantograph lacked the rotational direction.) It could be used for simulating non-centered or multi-point contact with two dimensional bodies in a planar environment; however, the authors did not make use of the rotational DOF—the study was strictly on single-point contact with a virtual wall. Audio responses consisted of a convolution-based modal synthesis. Because, similar to ACROE's approach, they used a single dedicated DSP for synthesizing haptic and audio feedback, they were able to create latency below 1 ms. Unlike Adelstein et al. [2], they aimed not to determine the JND of audio-haptic latency, but rather to verify that a positive or negative 2 ms delay would be a valid lower bound for human perceptual tolerance of synchronization latency. By varying the delay between haptic and audio responses they were able to determine that this is indeed the case. Specifically, they verified that there was no perceptual difference betwen 0.5 and 2 ms of delay, though the rate of decay of the sound had an effect on this perception. See Section 4.3 for a discussion on the topic of audio-haptic latency.

Feasel [29] did an unpublished but well-documented class project for Prof. Ming C. Lin involving a 3-DOF SensAble PHANTOM Omni device for plucking physically modeled

strings. He performed an analysis of guitar pick-string interaction and implemented a simplified model of it using the SensAble GHOST SDK. He reported that the interface is playable, though it takes some practice. The main problems he encountered were some difficulty in computing the device's handle orientation and the lack of rotational force feedback. The need for device torque is discussed further in Section 2.4.4.

FoleyAutomatic, created by van den Doel et al. [90], is a physically modeled audio-visual simulation of typical interactions used in the creation of sound effects. In particular, the authors modeled not only impact sounds, but also rolling and sliding interactions. The approach was to use modal resonance models for synthesis, and stimulate them using stochastic models of fine-grained interaction. They modeled a rock in a wok, which is a typical set-up for foley artists, as well as a screwdriver-bell interaction. The focus was on producing convincing sound effects by utilizing physically modeled environments, but there was little discussion on the importance of intuitive gestural control for such a system. There was, however, a brief mention of using a PHANTOM device to control the screwdriver, so that the edges of the bell could be felt.

Howard et al. [41] created Cymatic, another approach to physical modeling, partly inspired by CORDIS-ANIMA. A Cymatic instrument is constructed of masses and springs. A library of components is available, which are objects in 1, 2, 3, or more dimensions, and thus instruments that could not exist in reality can be constructed. As with CORDIS-ANIMA, virtual microphones can be placed on Cymatic masses so that the movement of a mass is turned into an audio signal. Tactile and force feedback can be provided through actuated joystick and mouse interfaces. Such an interface can be used to bow a Cymatic element. Alternatively, microphone input can be used to excite a mass, causing incident vibrations in connected masses.

2.2.3 Various musical haptic devices

Verplank et al. [91] created the Plank, a haptic controller specifically designed for *scanned synthesis* [93]. The idea behind scanned synthesis is to manipulate a two-dimensional curve representing a string at a slow rate, termed the "haptic rate" by Verplank, referring to the speed of human muscles. The curve is simultaneously played as the kernel of a wavetable synthesis at the audio rate. The Plank was built from an old hard drive motor, featured a Hall-effect sensor for position, and used a force-sensing resistor for determining pressure.

The user could hold the "plank" with his hand and move it left and right. Applying pressure to the FSR would create indentations or otherwise perturb the string model, creating changes in timbre. The motor could resist the left-right movement so as to create the impression of feeling bumps in the surface of the wave.[2]

Later, Verplank [92] enumerated a set of simple haptic musical gestures such as *pluck*, *ring*, *rub*, *bang*, *strike* and *squeeze.* He suggests that some of these can easily be achieve with only one degree of freedom. He used a 1-DOF device called the Force Stick to model them. It consisted simply of a short stick attached to a motor, which is capped with a force-sensing resistor. Data from the FSR is read by an Atmel microcontroller, which then changes a control voltage for the motor. Data is also sent from the microcontroller to a computer, where PureData is used for audio synthesis. Specifically, the effects he implemented are hitting a virtual wall, traversing a bumpy surface, plucking, friction, swinging a pendulum against a bell, and spinning a virtual wheel. He writes, "the surprise is that the 'best' haptics (precise, stable) may not be the most 'musical'." Additionally, he claims, "some sounds are impossible without the active force interaction."

Bennett et al. [7] examined the use of an electronically-controlled brake. As rotational movement of the apparatus controlled play head movement through a sound file, they used the audio output to modulate the brake. The effect was of being able to feel the resistance of the sound wave as it played. Interestingly, this experiment has some characteristics of force feedback, since it uses braking torque to resist user movement, but uses techniques usually exploited by vibrotactile feedback—that is, playback of the audio signal through an actuator. In an earlier study, Bennett [6] used the brake to simulate the hitting of a virtual drum kit.

2.2.4 Extending sonification of data through haptics

Hermann et al. [40] created the "audio-haptic ball" for sonic manipulation, a sort of "vibrating potato" which can be shaken, squeezed, hammered, moved and rotated to create different parametric changes. The intension was to use this device for navigating and controlling sonification of large amounts of data, to help "experience" the data in order to more easily find patterns and relationships within it. It is interesting in our context because the idea behind the ball was a deliberate attempt to avoid mapping data directly to

[2]Since the Plank involved a model of a touchable object, it could certainly be considered as a two-dimensional VMI.

sound parameters, but instead use the data as the model for a virtual object which is then interacted with via the device's sensors and vibrating actuator.

This sort of data representation is reminiscent of another use of the Moose proposed much earlier by Chafe and O'Modhrain [16] where the device was used to create a haptic representation of the similarity between different performances of the same piece of music. It showed that haptics can have a role not only in musical performance, but also in analysis and editing of musical representations.

2.2.5 Non-musical artistic uses for haptics

Baxter and Lin [5] created a haptically-enabled visual painting environment in which brush strokes and fluid viscosity are simulated accurately, and showed that such a system could be used to create actual artistic works.

There has been considerable interest in the use of 3D modeling for computer graphics [57]. For instance, SensAble [83] now offers two software products which utilize their haptic controllers to perform "clay-like" deformations of 3D models with force-feedback.[3]

Haptics have even been incorporated into a virtual hair styling salon [49, 96].

2.3 Non-tactual virtual instruments

For our purposes, non-tactual VMI may include any digital sound interface, usually represented with 3D graphics, which can be interacted with using an input device, and departs from the knobs and sliders often seen in the WIMP[4] screen-based interface paradigm.

In an early attempt to create musical instruments in virtual reality, Ng [65] extended the AVIARY virtual reality system to enable communication with an audio system. He created a 3D virtual recorder (woodwind instrument) that could be played by moving cubes over the finger holes. He envisioned a distributed virtual space in which musical performances could take place, with accurate spatialization of sound sources. He programmed AVIARY to use MIDI for triggering sound events on external synthesizer hardware. Mentioned also was the need for tactile stimulus, though haptic hardware was not available at that time. He unfortunately also did not have access to motion tracking hardware, and so his environment

[3] A full discussion of 3D "clay" modeling with haptics represents a large body of research and is out of the scope of this thesis.

[4] Windows, Icons, Menus, Pointers

was limited in practice to playback of pre-recorded files. Similarly, computer hardware at the time was not capable of producing high quality spacialization.

Choi et al. [17] created a "manifold interface" for exploring the parameters of an unpredictable audio circuit simulation. An image of the system is given in [4]. The manifold, representing continous transformations of circuit parameters mapped to the axes of a cube, is visually represented in three dimensions. The user can trace paths on the manifold which are then retraced to produce sound sequences. The interface is unique in that it presents a control space simultaneously with the phase space of the circuit being represented [4].

As mentioned in the previous chapter, Mulder [63] created two VMI simulations that used datagloves for input. They were the rubber sheet and the balloon. The user could push on the surfaces of these objects to create deformations, which created corresponding changes in sound characteristics. The simulations could also have certain nodes of the objects be connected to the positions of the thumb and index finger, so that the object could be stretched and twisted directly. The objects were modeled using spring-mass systems, which were constructed by fitting mathematical structures. For example, the curvature of the hand, computed from the positions of each finger relative to the angle of the palm, was applied to the balloon's ellipsoid curvature. An attempt was made to sonify the shape changes with associated sound qualities. The roundness of the balloon was mapped to the frequency modulation index, which provided a change in timbre. It is interesting to note Mulder's comments on user manipulation difficulties [63]:

> Touching the virtual object to move, rotate or shape the virtual object, required effective feedback of the contact between the hands and the virtual object. As no tactile feedback or force feedback was available, the performer had to rely on the visual feedback generated from the hand movements affecting the virtual object or the performer had to rely on changes in the sound. This method proved not so effective, probably because of a lack of suitable depth cues and because the absolute location of the virtual object had to be kinesthetically remembered (instead of being able to rely on tactile feedback when collision of the hand with object occurred) each time the hands moved without affecting the virtual object surface.

It is clear from this description that manipulation of a virtual object should be far more practical if a user has a tactual impression of its presence. It also shows that there may

be a need for stereoscopic viewing of the 3D scenario. However, it remains to be seen what effect the graphical display has on the VMI experience if there is an adequate haptic representation available.

Karjalainen and Mäki-Patola [45] created a virtual reality environment for musical interaction called EVE. Similar to Mulder's work, they used datagloves for input, with the addition of a stereoscopic display. They discuss the lack of tactual feedback, and suggest introducing real objects into the interface. Some examples of instruments they created using EVE are a virtual *air guitar*, in which the distance between the hands determines the pitch of a distorted Karplus-Strong guitar model; and a virtual xylophone, in which a user holds virtual mallets which can be used to hit virtual metal plates located in the air around the user. They report that users actually prefer to hold a real mallet for the same interaction, and that it seems to actually improve temporal accuracy. One might argue here that the use of real objects for controlling virtual interfaces begins to somewhat resemble the idea of *tangible computing*, such as the reacTable [44], for example.

Another consideration is that there is continuing interest in screen-based interfaces for music that depend on non-WIMP interaction and which incorporate manipulation of virtual objects in two dimensions. There is, for instance, currently a lot of interest in the intuitive gestural interfaces enabled by multi-touch display technology [22]. However, even for more usual mouse-based interfaces, groups like Ixi Software create atypical two-dimensional "toys" for musical interaction [51].[5]

2.4 Discussion

This section contains some reflections on aspects of haptic display put forward by the work discussed above. These issues were influential on the justification of using pen-based haptic displays, on the design of the implementation described in Chapter 4, and finally on what type of virtual instruments we might decide to construct for use with these devices.

2.4.1 Partially virtual instruments

One thing to note about the notion of an actuated keyboard, as compared to a pen- or glove-based haptic display, is that the main difference is in the choice to virtualize only part

[5]These are some examples, but a complete list of two-dimension interfaces and tangible computing interfaces is beyond the scope of this thesis.

of the instrument. An actuated keyboard virtualizes the piano *action*, equally enabling the emulation of any type of keyboard action, such as the grand piano, harpsichord, or organ [69]. Therefore, in a sense, each instance of an emulated piano action is a virtual instrument, whose physical controller is the set of actuated keys. However, despite its large number of degrees of freedom, each key being one, the lack of *integrality* [43] across these degrees of freedom is why the actuated keyboard would not necessarily be a choice manipulator for virtual instruments in general. We can assume that this is what led to the extensible design of the ERGOS device.

The human interfaces with the keys using his fingers and is able to feel their tactile properties such as the texture (or smoothness) of the material, as well as their particular shape, using the nerve endings in his skin. In comparison, in a true VMI, the physical interface takes the place of the *fingers* instead of the keys. A completely virtualized piano keyboard, where each key is an object in a virtual environment, could be actuated by the exact same response algorithm as the real motorized keyboard. The haptic interface would relay this information to the user through the response detected by the proxy object, which represents the hand or fingers in virtual space.

A human interacting with such a keyboard using a pen-based display may feel as though he is playing the piano with a long stick, and so the expected polyphony of the piano would likely make this a less-than-satisfying playing experience. On the other hand, an idealized glove-based force display that can related all tactile and kinesthetic sensations would be able to perfectly simulate a virtual piano. The notion that instruments can be virtualized at various locations in their mechanism—components of the virtual mechanism can be replaced with actuated physical manipulators or vice-versa—shows that the notion of a completely virtual instrument is not necessarily at odds with a partially virtual instrument: there is a dimension of continuity between physical controllers and their virtual counterparts.

An implication of this is that virtual instruments are inherently limited by the choice of physical input device for interaction with the environment. Aspects include not only the number degrees of freedom, but also the configuration of these degrees, including their direction and their integrality. Since we are exploring the use of pen-based devices, which tend to share these characteristics, this must be taken into account when designing our VMI's. We may very well be limited to instruments that are comfortable to manipulate with a "long stick." An alternative approach would be to first design our VMI and consider our requirements for an input device, but since it is more convenient to reprogram software

than to have our choice of ideal hardware solutions, the former approach is preferred here.

2.4.2 The importance of multi-dimensional configuration

From the previous section, it follows that the *number* of degrees of freedom is not enough information for determining the suitability of a haptic interface. The configuration of these DOF is equally important. In pen based displays, we tend to see them grouped around either 3-DOF or 6-DOF displays, with some exceptions. Within these groups, the dimensions are always orthogonal. This is, of course, due to the physics of the world in which we live. A point can be described by exactly three coordinates. Furthermore, that point, representing the location of a body, may be rotated around each of these axes, and thus its orientation can be described by exactly three rotations with a known order. These are the point's Euler angles, often represented in order-independent fashion as a matrix or quaternion. Being able to then manipulate this point to any position or orientation creates a certain *generality* for the gestural interface: a 3D or 6D *point* can be imagined as representing any object in a space. In comparison, a 6-key piano keyboard also has 6 DOF, but since they are separate and their configuration is not orthogonal, they do not easily represent a point in space in a natural way. Trying to do so would be similar to creating a beautiful painting using an Etch-A-Sketch. Couturier [19] would say that a pen-based display has a high *degree of compatibility* with the physical world.

This theory has indeed been tested in the context of control. Jacob et al. [43] tested users on a three-dimensional target-finding task. Subjects controlled an object using either a Polhemus position sensor or using the mouse for x and y position and an on-screen slider for *depth*. The experiment showed a marked improvement in performance when "the structure of the perceptual space of a graphical interaction task mirrors that of the control space of the input device."

Interestingly, it is possible that the presence of haptic feedback may have some effect on this principle. Florens et al. [31] mentions that they tested users on two control structures, or *ergonomies*, for manipulating the rattle model. The first was a separated control where users could affect x position with one hand and y position with the other. The second was an integrated 2D stick. They found, qualitatively, that there was no obvious difference in performance between the two situations. Further, they claimed that users found the first situation was more accurate and easier to exert precise control. However, when

haptic feedback was removed, the first situation became "absolutely ineffective," and "uncontrollable." Unfortunately, no comments were available about the effects on the second situation.

More work on this effect would be quite interesting, but, despite the reports of better accuracy in the separable situation, we find that pen-based control, where all dimensions are integrated to the fullest degree, is an adequate and useful mode of interaction for the general case.

2.4.3 Physical accuracy

Sounding objects

Restricted to the detection of impact collisions, the level of control in a simulated instrument may be limited, like the piano hammer [33], to velocity and timing, perhaps with the addition of contact location; it is no wonder that a performer would consider micro-scale sliding and rolling interactions to be critical for generating subtle and expressive sounds from percussive instruments, simulated or real [90]. While this thesis is not expressly concerned with sound synthesis methods, it should be noted that acoustic modeling of various kinds is becoming quite popular in sound design for 3D simulations. A good overview of available physical modeling techniques is given by Karjalainen and Mäki-Patola [45]. The use of rigid-body simulation for controlling physically modeled sound has been explored previously [70], and tends to often be targeted towards sound effects for 3D animation or video games. The reason is mostly self-explanatory: graphical simulation of a real object in virtual space seems to go hand in hand with its acoustical counterpart.

For Luciani et al. [48], the physically consistant preservation of energy from gesture to sound is of utmost importance. However, the reader should keep in mind that in designing VMI's based on Mulder's view, we are not directly concerned with this. Rather, the goal is to create good *controllers* for sounds that may or may not be related to their physical properties—that is, we are not trying to create *sounding objects* [79]. Physically modeled sounds can, of course, provide parametric spaces that map well to the output of a virtual controller, and so the use of acoustic modeling is a good avenue for sound design in this context, but it is just as valid from the perspective of virtual controller design to map to sounds that are entirely disconnected from reality, as long as they are rich in timbral dimension and the gestural control can be intuitive to the performer. For instance, in many

cases we are looking to create a virtual controller for a sound synthesis algorithm that has already been implemented. Thus, the design of the controller is based on the sound, rather than the converse.

Geometric vs. dynamics-based simulation

A point of view derived from the energy-preserving *ergotic* scenario described in Luciani et al. [48] is that a simulation based on preserving the dynamics of an interaction is more important than keeping the exact geometric nature of the physical objects being simulated. The bowing simulator they describe makes no effort to reproduce the shape of a violin, but rather only ensures that the physical dynamics of the bow-string interaction are accurate. They describe how this simplified the physical model and allowed players to forget about physics while playing the instrument.

While this is an interesting observation, in this work I argue that maintaining a geometric model of the interaction certainly could not detract from the experience, and moreso that it might provide a more concrete experience both for the instrument designer and for the player. If we take the view, as described in the introductory chapter, that a virtual instrument is merely an extra layer of mapping based on a physical metaphor of manipulating a virtual object, then being able to visually or haptically construct an internal image of the control surface is what allows the player to maintain this metaphor.

Vibrating bodies

While vibrotactile stimulation—the feedback that reminds a player that his instrument is "alive"—is certainly an important aspect of haptic feedback in gestural control, this thesis is more concerned with how force feedback may be used to provide a tactual image of a metaphorical controller. It is important to realize, then, that tactile feedback is in some ways inseparable from force-based display. This is because a force display must include some form of friction model for the tactual image to be usable. Without friction, a virtual object tends to feel "slippery" and can be difficult to manipulate. The presence of friction is connected to the concept of texture, since the type of friction exerted by lateral movement along an object's surface will give rise to an image of the object's material properties. Coulomb friction tends to feel "rubbery", for example.

At this time, I am explicitly not concerned with simulating perfectly the continuity

between frictional energy and acoustic vibration, since I have chosen to concentrate on force feedback rather than vibrotactile feedback. Thus, the objects simulated by the environment described in Chapter 4 have frictional properties for the purpose of making them easier to interact with, but the acoustic vibration simulated by the external audio synthesis does not feed back into the haptic environment. It may be possible to find a solution for this within the described architecture, but for the moment this is left for future work. This topic is further discussed in Section 5.2.4.

2.4.4 Rotational feedback

Feasel's experience with simulating the guitar pick is exemplary of the currently underestimated need for torque feedback in haptic devices. A large number of real-world interactions make contact at points that are not centered on the location of the hand or fingertips. In interactions involving tools, the fingers typically grasp the tool at one point, while contact is made with an object at another point. Any force vector creates torque around the point where the users grasps the tool, which is felt in the wrist. Musical examples include picking a string, hitting with a drumstick, and bowing. The vBow, for example, benefitted from rotational feedback, which allowed it to render the positions of virtual strings.

Torque can also be useful for moving and orienting objects that have inertial properties in a virtual space. In pen-based haptic displays in particular, the point interaction does not allow tracking of the fingers, and so simulations are most likely to involve tools grasped by the whole hand. Therefore, torque display is definitely an interesting avenue for musical interaction research. There are, unfortunately, fewer available haptic devices which have the capability to exert rotational force, and these carry higher price tags. This only makes it more important to show that this may be a useful feature for haptic devices, so that the industry can become interested in producing more affordable 6-DOF devices.

2.5 Summary

This chapter has given a history of virtual musical instruments and the use of force-feedback for musical purposes. Several issues relating to the implementation described in this thesis were discussed. Chapter 3 will provide an overview of currently available software development solutions for haptic displays.

Chapter 3

Software Survey

The construction of a haptic virtual environment requires the use of several types of software. Firstly, drivers for communicating with haptic hardware are required. Secondly, software that implements collision detection for haptic simulation is needed. Typically this software will maintain a *scene graph* which tracks the objects in the environment in a hierarchical manner. Thirdly, for creating dynamic behaviour, a physical dynamics engine, also called a rigid body simulator, must be used, which is responsible for tracking state information about each object's mass and velocity, ensuring that they behave realistically when collisions occur. Support for graphical rendering of the environment is also needed. Lastly, distributed virtual reality software to enable collaborative interaction may be desired, but that is beyond our scope here.

There is a certain amount of overlap between these requirements: there are several libraries which provide access to haptic devices but also have functions to perform collision detection, maintain the scene graph, and display it on the screen. However, most of these libraries do not provide physical dynamics, and so a static environment is assumed. On the other hand, physical dynamics libraries typically have their own copy of the scene graph which must be managed to ensure that it is synchronized with the visual-haptic scene. It should be noted that a physical dynamics library in fact contains many of the same classes of algorithms as a haptic library, since haptics and physics share similar collision detection requirements. However, in a "static" haptic environment, typically the collision detection is required only for the proxy object representing the haptic device's end effector. This is more often than not assumed to be a sphere, or even a sphere representing a point, which

greatly simplifies collision detection.

Unfortunately, it also inherently limits interaction to three degrees of freedom. In contrast, physical dynamics engines usually provide 6-DOF physics, since objects often collide at positions away from their centers of mass. In principle, a physical dynamics engine could be turned into a haptics library by having one object represent the haptic device's end effector, as discussed by DeFanti et al. [25]. In practise however, some 6-DOF physical simulations may have difficulty meeting the real-time requirements of a 1 KHz haptic servoloop when a moderate number of objects are in the scene. This is changing as the available processing power continues to increase, but no offering which performs 6-DOF physical dynamics and incorporates haptic support is yet available. Section 4.4 will describe one solution to this problem, in which static haptic processing is performed in parallel with slower 6-DOF dynamic physics processing.

This chapter provides an overview of several software environments either commercially or freely available for creating force feedback-enabled simulations. A short overview of the capabilities and availability of each is included, as well as a brief description of workflow in the environment. We also describe the physical dynamics environment chosen for the work described in the next chapter.

3.1 Haptics software

3.1.1 CHAI 3D

CHAI 3D[1], which is pronounced like Chai tea and stands for Computer Haptics & Active Interfaces, was created by Conti et al. [18] at Stanford University. It is a C++ class framework which is able to represent a scenegraph composed of both mesh-based objects, which can be loaded from disk in the 3DS format, and objects defined by implicit functions [81], such as spheres and torii. The library makes use of C++ inheritance to abstract the device drivers, so that various haptic devices with differing drivers and I/O interfaces can be supported by the same source code. It supports 3-DOF interaction, though the latest version is able to detect mesh-on-mesh collisions. At the time of this writing, it supports devices from SensAble, MPB Technologies, and Force Dimension, as well as the ServoToGo and Sensoray digital I/O boards, which are currently popular choices for robotics development.

[1] CHAI was chosen as the basis for the software discussed in Chapter 4.

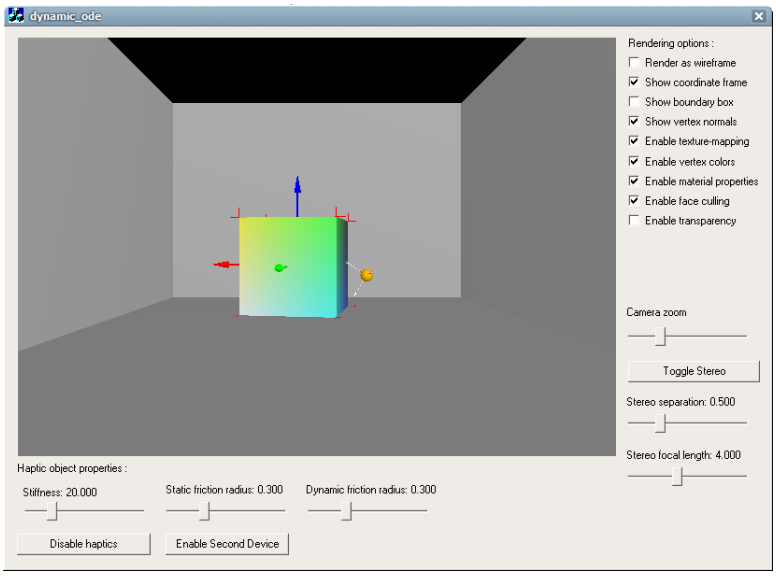


Fig. 3.1 A CHAI 3D demonstration application showing how to use it in conjunction with the OpenDynamicsEngine. The cube can be pushed around using the haptic device, and it bounces off the walls. CHAI can be used to visualize object axes and surface normals, as shown here.

CHAI 3D is free to download and is open-source software covered by the GNU General Public License version 2 (GPLv2). It has been developed on Microsoft Windows but also runs on Linux for devices which provide drivers, and can also be pursuaded to run on Apple OS X as well. Haptic object "material" properties such as stiffness and friction coefficients can be tuned, and two types of collision detection optimizations are available. For graphics, it supports various colour properties, transparency, and environment mapping which are drawn using OpenGL. For debugging and analysis purposes, internal structures such as collision trees, normal vectors, force vectors, and local object axes can be visualized.

3.1.2 Haptik Library

The Haptik Library [24], developed by the Siena Robotics and Systems Lab (SIRSLab), provides a device-independent interface to pen-based haptic displays. Unlike many other software packages discussed in this section, Haptik does not provide scenegraph tracking or any built-in rendering algorithms. However, it uses a component-based approach for accessing haptic hardware. "Component-based" refers to a specific paradigm for handling communication between software modules, which is also seen in Microsoft software (COM),

Mozilla (XPCOM), and CORBA. Instead of providing device abstraction by using C++ inheritance and polymorphism, as in CHAI for example, devices are accessed through a well-defined, language-independant *interface*. Interfaces guarantee forward- and backward-compatibility on the binary level, so that software components can be upgraded without causing errors and without requiring recompilation. New features are added by creating new interfaces, and components can continue to support old interfaces while simultaneously providing new ones. Since component interfaces allow complete independance from implementation details, a particular implementation can provide other services such as recording and playing back data, device usage over a network, or mouse-based emulation of haptic hardware for debugging purposes.

By providing a generic interface called IHaptikDevice representing any 6-DOF haptic device, the library allows completely hardware-independant development. (To be clear, 6-DOF collision detection and dynamics are *not* provided.) Components implementing this interface can contain the interface to the hardware's actual driver. Currently Haptik has implementations for devices from Force Dimension, SensAble, and MPB Technologies. The Novint Falcon is mentioned in the article, so presumably it will be supported once it is released. Haptik provides either a callback-based or polling architecture for running the high-priority haptic rendering thread. The paper mentions that Haptik has been used as a device driver in both the H3D and CHAI scenegraph packages. Some software examples are given implementing very simple haptic effects in only four or five lines of C++ code. It runs on Microsoft Windows and Linux.

3.1.3 osgHaptics

Maintained by Umeå University's VRlab, osgHaptics [3] is an extension to the OpenSceneGraph project [74]. The OSG project is an open-source C++ library for tracking scenes containing 3D objects. It has been used in projects such as games, virtual reality applications, and visualization. osgHaptics uses the OpenHaptics toolkit from SensAble, (see Section 3.1.7,) to allow haptic rendering of objects represented in the scene graph. Haptic materials may have stiffness, damping, and friction properties. The software relies on OpenHaptics to perform collision detection and to communicate with the haptic device. As such, osgHaptics acts as a bridge between an OSG scene and a SensAble haptic device. Software using OSG for visualization may find it a convenient way to allow haptic inter-

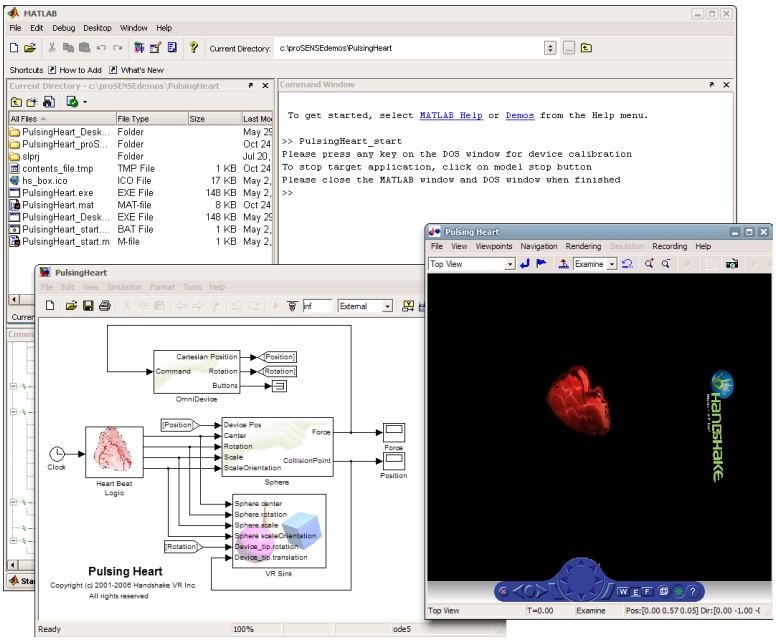


Fig. 3.2 A proSENSE demonstration application. The user can feel the virtual heart pulsing when the haptic proxy is placed on its surface. The Simulink graph used to create the demonstration is shown here. The scene is described by a VRML file.

action, however the toolkit is limited to devices supported by OpenHaptics. An additional consideration is that OSG is a particularly large library, and not all projects will find it convenient to use for simple tasks.

3.1.4 proSENSE

An offering from Handshake VR, based in Waterloo, Ontario, the proSENSE Virtual Touch Toolbox is a set of Simulink function blocks for Mathwork's popular Matlab scientific computing software that allow the rendering of haptic effects and communication with several haptic devices. It works in cooperation with several other Matlab toolkits, including the Virtual Reality Toolkit and the Real-Time Workshop (RTW). Function blocks are linked to each other graphically in a Simulink window, allowing for a visual style of programming. The VR Toolkit is used to provide a graphical environment based on the VRML file format, which is also synchronized with the haptic model in Simulink. The RTW is used to generate C code and compile it into an executable, which is then run in parallel with the graphical simulation. The real-time portion of the code may run on another computer,

using a network connection for synchronization. The generated C code is also linked with proprietary pre-compiled object code.

Interfaces are available for devices from SensAble, Force Dimension, Quanser, and MPB Technologies, as well as with Microsoft Windows-compatible input and force-feedback devices such as joysticks and mice. Modules are available for simulating various simple objects including boxes, cones, cylinders, spheres, elevation maps, and mesh-based objects. Objects have properties such as size, stiffness, and friction parameters. Friction is modeled using Coulomb friction with or without stiction. There are also modules for creating springs, inertia, and damping. Additionally, several functions are available for performing time delay compensation for operation over a network, including Handshake's TiDeC algorithm which is able to guarantee stability over long-delay connections. It includes limited sound support in the form of a module which can play back a sound file based on an event trigger.

This toolkit would be best recommended for users who are familiar with the Simulink environment and wish to incorporate haptic feedback into Matlab-based VR simulations. For example, it may be useful to those wishing to use haptics for exploring data sets preprocessed in Matlab. While sound support is mostly lacking, Simulink can be extended by a C programming interface, so that communication with a more interesting audio engine might be established. There has been some effort by Christian Frisson and David Birnbaum of McGill's Input Devices and Musical Interaction Laboratory to create an Open Sound Control module for communicating with audio synthesis software, however this software remains thus far unpublished.

One pitfall to bear in mind is that there is a different Simulink block for each supported device, and no abstraction of the interface is provided. This means that changing hardware will require some minimal amount of reprogramming.

3.1.5 Reachin API

The Reachin API [77], an offering from Reachin Technologies of Stockholm, Sweden, is a complete haptics development framework. It was the first haptics/graphics framework to support multiple devices [18]. It supports scenegraph management using the VRML file format. Elements of the VRML scene can be given haptic properties, so that they can be touched and manipulated with a haptic device. Object properties can be scripted using the Python programming language. Objects can be given deformable properties, so

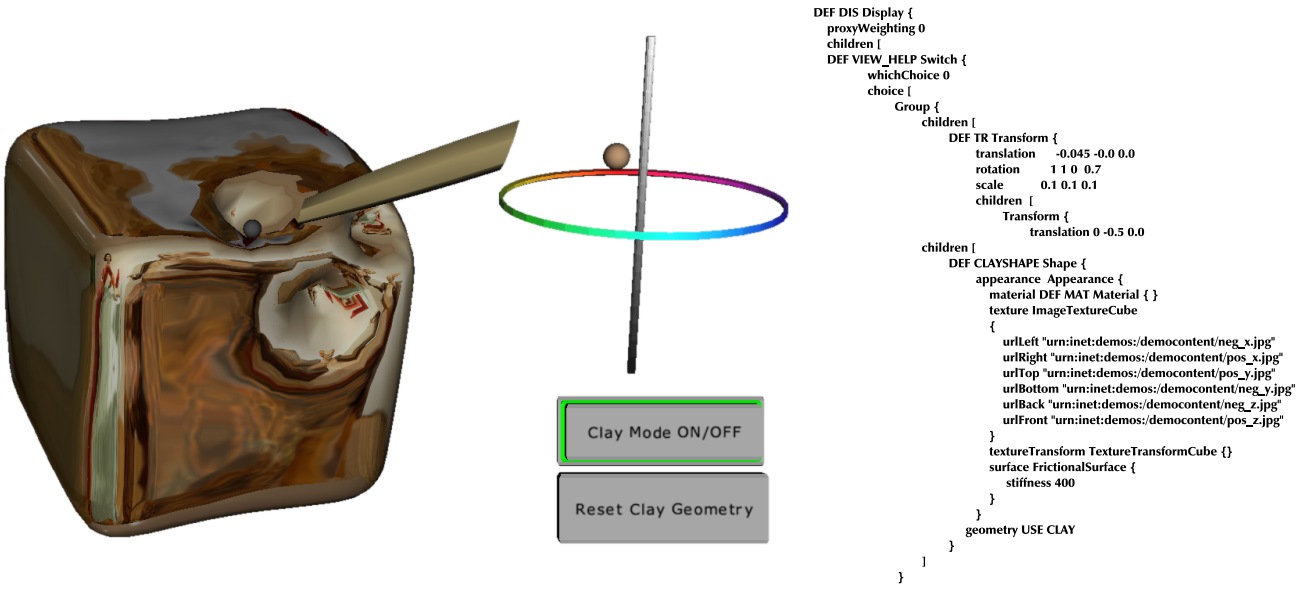


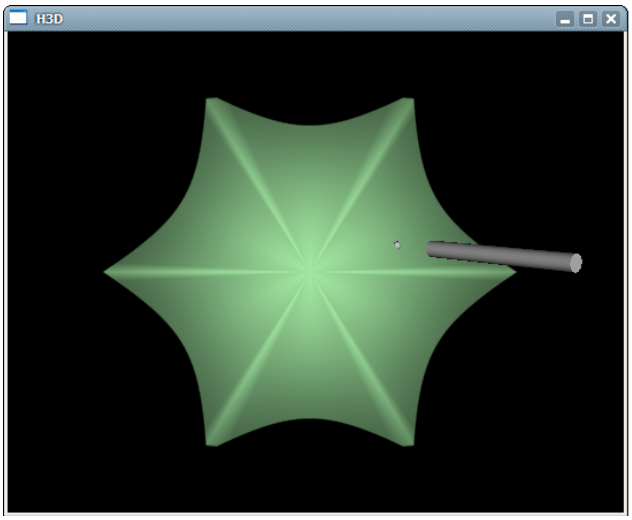
Fig. 3.3 A Reachin API demonstration application. A portion of the VRML code used to define this environment is shown on the right-hand side. The object can be deformed like a clay model. Some user-interface elements are shown: two push-buttons which have a haptic "click", and a colour wheel.

that they can become elastic or clay-like, and bump-mapping techniques can be used for both graphical and haptic rendering of bitmap-based textures. Reachin supports several GUI-style interaction elements, such as push-buttons and color selectors, which can be manipulated with the haptic device.

Reachin supports devices from Force Dimension, SensAble, and MPB Technologies. It has support for multiple devices, enabling a collaborative environment. Stereoscopic views are also supported. Resources such as scenes and materials can be specified using Internet-aware Universal Resource Names (URN), a scheme allowing the unique identification of a resource which may be located remotely. An account of incorporating haptic support through Reachin into a virtual reality system can be found in [1].

3.1.6 H3D

H3D is an open-source offering from SenseGraphics. It is dual licensed under the GNU General Public License as well as a commercial license. It runs on Microsoft Windows, Linux, and Apple OS X, and is based around the X3D file format, an XML-based ISO-standard language for describing 3D scenes and objects which is often thought of as the




```
<?xml version="1.0" encoding="UTF-8"?>
<Group>
   <FitToBoxTransform boxCenter="0 0 0" boxSize="0.3 0.3 0.3"
   uniformScalingOnly="true" active="true">
     <Shape>
       <Appearance>
        <Material diffuseColor="0.7 1 0.7"/>
        <SmoothSurface />
       </Appearance>
       <SuperShape DEF="SS" resolution="128"
                   ss1_m="6" ss1_a="1" ss1_b="1"
                   ss1_n1="1" ss1_n2="1" ss1_n3="1"
                   ss2_m="3" ss2_a="1" ss2_b="1"
                   ss2_n1="1" ss2_n2="1" ss2_n3="1"  />
     </Shape>
   </FitToBoxTransform>
</Group>
```

Fig. 3.4 An H3D demonstration application. The X3D file which described the scene is shown on the right. The object can be touched and feels smooth.

successor to VRML. It can be programmed directly with C++, but also has a binding for the Python dynamic programming language. Similar to osgHaptics, it depends on OpenHaptics to perform collision detection and haptic rendering, and thus a SensAble license is required for use. However, since it contains a driver using the Haptik Library as a back-end, it is possible in practice to use it with non-SensAble devices, as long as the OpenHaptics library is available. Built-in collision detection may be added in future revisions. In addition to X3D meshes, it supports primitives such as boxes, cylinders, and cones. Like Reachin, scenes and materials can be specified using Internet-aware URN's. Several advanced graphics techniques can be used, such as shaders, textures, and stereo rendering.

A demonstration application which comes with H3D can be seen in Figure 3.4.

3.1.7 OpenHaptics

OpenHaptics is the toolkit distributed with SensAble's PHANTOM haptic devices. It is a successor to their previous GHOST SDK. GHOST was the first haptics programming framework to allow high-level definition of scene graph elements which are rendered without the need for user-defined algorithms. Many research users bypassed this functionality for the sake of exploring new rendering methods. As a reaction, SensAble created OpenHaptics as two libraries, a "device-level" library called HD and a higher-level library called HL.

The HD library provides a real-time callback which runs at the haptic rate. Functions are provided for reading the device's position and writing values to the device's motors in either Cartesian or joint-space coordinates. The HL library is a set of functions modeled after the design of the OpenGL graphics API. It provides function calls that delimit the start and end of a frame render, which can be inserted in a typical OpenGL drawing function. Between these two function calls, OpenGL calls are detected and used to generate the haptic scene. Thus, OpenHaptics makes it possible to re-use graphical rendering routines to define the haptic scene, meaning that rendering code need only be written once. In addition to frame delimiting, functions are available for specifying haptic properties such as friction and stiffness.

It supports two methods for haptic display. The first is the "feedback buffer" technique, which uses OpenGL data structures to extract the geometric properties of the scene. This is similar in principle to most scene graph libraries described in this chapter. The second technique, called "adaptive viewport", uses the graphics card to render a height map of the current scene, which is retrieved from the graphics hardware's depth buffer. The height map is then used to calculate collision with the haptic proxy. Since this technique makes use of the graphics hardware, it is able to render more triangles and much more complex scene geometry. Some drawbacks are that strange effects may be produced by concave surfaces or if the haptic proxy moves to quickly. The documentation claims that for most purposes a height map rendering frequency of 30 to 60 Hz is sufficient for most applications.

OpenHaptics, despite the name, is a closed-source library and supports only SensAble devices. It runs on Windows and Linux, the latter being available separately.

3.1.8 ACROE CORDIS-ANIMA

CORDIS-ANIMA, created by Cadoz et al. [14] for their research in physical modelling for use with multi-sensory systems, is designed to describe fundamentally physically-driven simulations. I include it in the category of Haptics software because it describes a scene composed of objects which can be touched with a haptic interface, however it is more accurate to say that it is in fact *particle-based* rather than dealing with object primitives. Elements in CORDIS-ANIMA are one of four basic types: the first three are *mass*, *spring*, and *friction* elements, which can construct what they term "linear matter." The fourth element introduces non-linear relationships and is called a *conditional link*. Objects are

constructed by connecting masses to each other through springs. The objects then behave in a physical manner, allowing deformations to occur. The object deformations, through which oscillation arises, can be used to generate sound. Simultaneously, a specially-designed haptic interface, now commercially available through ERGOS Technologies, can be used to interact with the scene. Thus, the same physical model is used to produce audio, graphic and haptic feedback. In the ERGOS system, the model is run on a dedicated DSP board which is connected to the haptic device. The DSP is embedded in a computer through a PCI interface, and the PC is used to general graphical and audio feedback.

3.1.9 Summary

Table 3.1 summarizes the capabilities and licenses of the toolkits described above. It shows a few trends: though there is some cross-platform support—particularly in the open-source offerings—the best device support is for Microsoft Windows, with particularly little attention given to Apple OS X. This is interesting since audio research, which we are concerned with here, is one area in which OS X has attained a high level of popularity. Most toolkits consist of C++ frameworks, although some alternative languages and text-based description formats can be used.

Not present in the table is that sound support is only explicitly discussed by the proSENSE documentation, with the exception of CORDIS-ANIMA for which sound is as important as the other sensory modes. Several libraries do come with examples of how to use sound file play-back routines in conjunction with haptics, but these capabilities are not directly supported by the library, and often rely on operating-system specifics such as Microsoft's DirectX.

Most libraries provide some support for various commercial hardware solutions, though there are cases where only SensAble devices are supported. Again, CORDIS-ANIMA is the exception here, where the language is compiled for a special DSP card which executes the program independantly from the PC, with a direct interface to a particular hardware device.

3.2 Physical dynamics engines

There are many software libraries available both open-source and proprietary that provide routines for calculating physical dynamics. A complete survey of physical dynamics software

Name	Devices	Scene Graph	Mesh/Scene Support	DOF	Graphics	Dynamics	Operating System	Programming Language	License
CHAI 3D	SensAble Force Dimension MPB Technologies	Yes	3DS	3	OpenGL	No	Windows, Linux	C++	GPL
Haptik Library	SensAble Force Dimension MPB Technologies	No	N/A	N/A	N/A	N/A	Windows, Linux	C++, Java	GPL / MIT
osgHaptics	SensAble	Yes	Many available	3	OpenGL	No	Windows	C++	LGPL
proSENSE	SensAble Force Dimension Quanser MPB Technologies	Yes	VRML	3	VR Toolkit	Yes	Windows	Matlab/ Simulink	Proprietary
Reachin API	SensAble Force Dimension MPB Technologies	Yes	VRML	3	OpenGL	For UI elements	Windows	Python, extended by C++	Proprietary
H3D	SensAble (required) Force Dimension MPB Technologies	Yes	X3D	3	OpenGL	No	Windows, Linux, OS X	C++, Python	GPL + commercial
OpenHaptics	SensAble	Yes	Application-provided	3	OpenGL	No	Windows, Linux	C / C++	Proprietary
CORDIS-ANIMA	ACROE ERGOS	Yes	Application-provided	Variable	?	Yes	?	CORDIS-ANIMA	Proprietary

Table 3.1 Haptic toolkits

is beyond the scope of this thesis. However, the Open Dynamics Engine (ODE), which was chosen for this project, is described here briefly. It was chosen because it is free and open-source, can run under real-time constraints, and was adequate for our purposes. An in-depth comparison between ODE and several other physical dynamics engines can be found in [87].

3.2.1 The Open Dynamics Engine

Created by Smith [88], ODE is an open-source physical modelling engine which can be used in conjunction with a graphics library to provide realistic physics for 3D environments. It supports several geometries, such as prisms, spheres, cylinders, planes, and triangle meshes. Object geometries, called "geoms", are used for collision detection and can be attached to rigid bodies which track physical properties such as position, orientation, mass, and velocity. This separation between collision detection and the physical environment provides powerful ways to create interesting dynamics. For example, multiple geoms can be attached to a single body to create compound objects, or geoms can be dissociated from their body in order to remove them from the simulation. Additionally, body movement can be constrained in a number of ways by creating *joints* between them. Many joint types are available, such as ball joints, hinges, universal joints, and sliding joints. Bodies can be constrained relative to each other or relative to the global coordinate system. This way, physical mechanisms can be created. ODE is cross-platform, with no dependancies on a

particular graphics library.

3.3 Summary

This chapter described several currently available solutions for haptics programming. A common theme throughout has been the use of a programming language or description file for defining the scene contents, which is then compiled or interpreted by the software. In contrast, audio environments such as PureData [75] or Max/MSP [20] allow the dynamic creation and destruction of audio units during run-time. This is a paradigm which has proven useful to researchers and artists in sound and visual media, who enjoy the flexibility of modifying algorithms as they are executed, and appreciate the user-friendly approach of visual programming. The technical knowledge currently required to create physically realistic virtual environments and to take advantage of the haptic channel makes these ideas either unaccessible or at least inconvient for this audience.

In the next chapter, we will see an approach to haptics which allows run-time experimentation with objects in a virtual environment, as well as communication with third-party audio software running in parallel on the same computer or elsewhere on a network.

Chapter 4

A Simulation Server for Virtual Musical Instruments

4.1 Introduction

In the previous chapter, I described several software solutions for creating haptic simulations. Unfortunately, with the notable exception of CORDIS-ANIMA, none of these solutions provides very interesting support for audio. Additionally, unlike audio research software, they do not generally allow dynamic modification of the simulation at run-time, and require a certain level of programming ability. They also do not make it easy to create physically dynamic environments with objects that react to each other, since rigid body simulation is typically only available as a separate library.

For the purpose of creating virtual musical instruments, I am interested in incorporating all these attributes into an easy-to-use system for creating and experimenting with haptic interfaces for music. Ideally, a haptic virtual controller should interface with previously existing audio software instead of trying to re-invent that particular wheel, and should be accessible to non-programmers so that it is a useful tool for musicians and researchers in psychology, music technology, and related areas.

4.2 Case study: video integration in an audio system

As an example of an audio system that has been expanded with visual capabilities, we can look at PureData's GEM set of externals, written by Danks [21]. GEM provides a viewport

for an OpenGL environment, and several objects for controlling what is drawn into it. It supports several 2D and 3D primitives such as squares, circles, cubes, spheres, and cones. These shapes can be texture-mapped with images or video. It can also perform certain image processing tasks, like alpha masking or convolving two images.

PureData processes data in two ways: firstly, it has *control* data, which is processed asynchronously. Whenever an event occurs, like an incoming MIDI note for example, it is propagated immediately through the network of connected objects. Delay and metronome objects can be used to introduce timed events. Secondly, it has *audio* or DSP data, which flows continuously through a separate network of objects, updating audio hardware buffers in real-time. The two networks are connected by DSP objects which can accept control data. Therefore, DSP parameters are updated whenever control processing can occur, slotted between two audio cycles.

GEM works at the control data level. When the screen must be updated, a GEM object called "gemhead" issues a command to be passed down the chain, potentially through geometric transformations, until it reaches a final drawing object. The timing for these operations is not considered as critical as for the audio chain; though it is not desirable for video output to glitch or pause, it is able to run at a lower priority than audio because the timing interval between frames is slower, as described in the next section. However, the GEM documentation does warn that CPU-hungry image processing may cause playback glitches in the audio stream, which shows that potential problems can occur when a processing task runs in synchrony with a real-time audio stream.

The system described in this chapter began life as an idea for a "GEM for haptics." It quickly became apparent, however, that it would be more fruitful to run a haptics simulation in a separate process. The reasons are several: unlike video, haptics has real-time requirements that are more strict than those for audio, making the addition of a haptics DSP chain to PureData require non-trivial changes to its core engine. Additionally, because of the potential for CPU hogging, it is useful to have the option of running a haptics simulation on a separate computer instead of within the same process as an audio/visual engine. Finally, we would like to create a haptics system that is useful for PureData but also for other audio software, and creating PureData-specific extensions would not have allowed this as easily as taking advantage of a communication protocol shared by several audio systems.

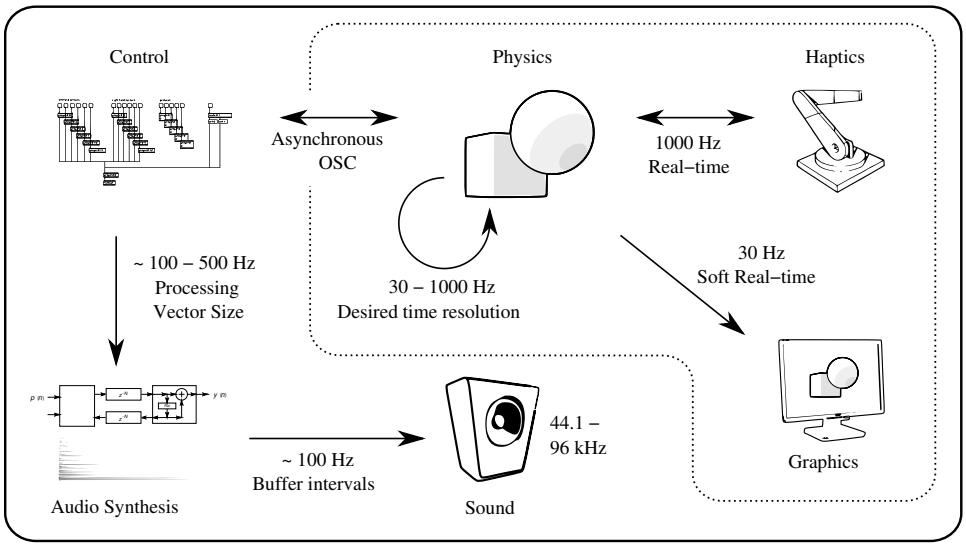


Fig. 4.1 Each part of the system has different timing requirements and runs independently. The process boundary is delineated by the dotted line. Each part of the simulation process, within this line, is a separate thread, while the audio process can be any software which is OSC compatible.

4.3 Latency requirements

A rendering system for a multi-modal display is inherently separable by each of its sensory modes. While some common properties can be shared, each mode has different requirements regarding timing and data throughput. For example, while control changes should be apparent in an audio stream within 10 ms or less for a satisfying user experience [52], visual displays usually update at about 30 Hz, meaning that control changes are allowed up to 33 ms to be received and processed.

In contrast, force-feedback haptics requires the total latency be 1 ms or less. This is because input and output are directly coupled: the user is part of a closed system. The "display" depends entirely on the user's movement, and reactions to position changes must be as instantaneous as possible in order to render the feel of a hard surface. It has been previously found that between a 500 Hz and 1 kHz update rate must be maintained for a good user experience [59]. These timing requirements and their inter-relationships are shown in Figure 4.1.

Distinct from the haptic closed-loop rate is the audio-haptic latency. This is the time between haptic events and a resulting sound. Adelstein et al. [2] investigated this latency.

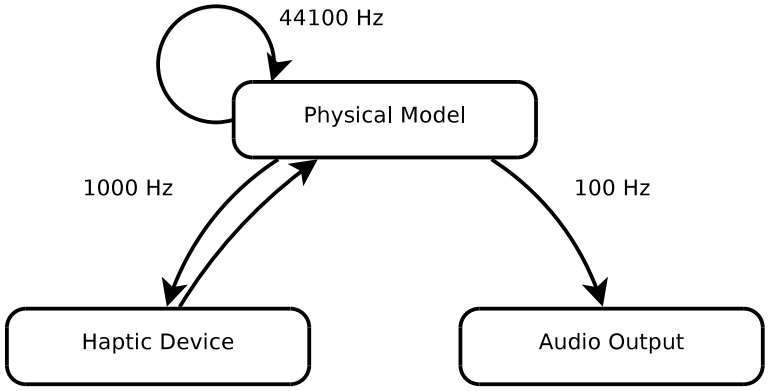


Fig. 4.2 A synchronous system architecture in which audio and haptic sensory cues are derived from the same physical model.

They used an accelerometer attached to a hammer to detect taps, and converted the signal to MIDI to generate audio events. Subjects wore headphones so that they felt the hammer independantly from hearing the results. With this apparatus, they showed that the measured just-noticeable difference (JND) for audio-haptic latency is about 24 ms, with a 2.2 ms standard error. This has implications for the time allowed for information to flow from a haptic simulation to an audio process, though evidence suggests that audio-haptic timing maybe be more strict for active (motorized) haptic devices as opposed to passive apparatus [26].

4.4 System architecture

As such, there are choices to make in terms of how the system architecture will take these differences into account. In the ideal case, a single fast processor could be used to perform all operations in synchrony. A single simulation of the environment would be used to derive haptic, graphic, and audio feedback, as shown in Figure 4.2. In practice, running a complete simulation at the audio rate will quickly exceed computing power for a moderate number of objects. An oft-used optimization is to allow the physically vibrating bodies to be modeled in a more efficient manner, using techniques such as modal synthesis or waveguide modeling, while the macro interactions between objects are computed at haptic rates.

When the two parts of such an algorithm are run synchronously on a single processor, this architecture usually provides minimal latency between each sensory mode. It is

used, for example, by the CORDIS-ANIMA system which makes use of a dedicated DSP board and attempts to create a very high-fidelity user experience. Additionally, since a realistic physical model is used for both gesture and sound, it creates an energy-preserving continuum between instrumental gesture and the resulting audio output, which is ideal for "playing" virtual physical objects intuitively [48]. As a second example, this architecture was also used for the AHI project [26], running interrupt-driven haptics and sound on a DSP, in order to measure a lower bound for haptic-audio latency.

However, from the point of view of creating virtual gestural controllers for existing synthesis systems, a disadvantage of using a single physical model for all sensory modes is that it imposes restrictions on the chosen audio synthesis model. Essentially, physical modeling is the only acceptable audio synthesis in this paradigm, ignoring that we may wish to control, for example, FM synthesis, etc. In fact, since the physical model takes care of audio computation, it is not clear how to best interface such a system with existing audio software running on a normal desktop computer.

Additionally, there is a tendancy in uses of the synchronous model to use dedicated DSP hardware to provide tight control over timing and synchronization. Since most commercial haptic devices are used with desktop computers, and because VMI designers would prefer to work with audio software they are already comfortable with, a solution is required which can take advantage of pre-existing hardware and software. An implication of using standard PC hardware is that there is an upper-bound on achievable data rates, since most operating systems allow timing *just* within 1 ms, but it also means that the computational architecture can be extremely flexible—for example, taking advantage of the growing popularity of multi-core processors [89], or even high-bandwidth local networking to allow a tiered approach [54].

Therefore, an alternative to the strictly synchronous design discussed above is to consider each sensory mode independently, running in separate processes with asynchronous timing, communicating events and continuous information to each other. This architecture stresses that it is not the *synchroneity* that matters, so much as the *latency* between each sensory mode that must remain below the level of human perception. This is the approach I have taken with the system described here, seen in Figure 4.3.

Each module of the multi-modal simulation runs separately, at different rates, using either shared memory or some other communication method to stay synchronized. The physics model is updated at about 100 Hz, though this is variable. The haptic device

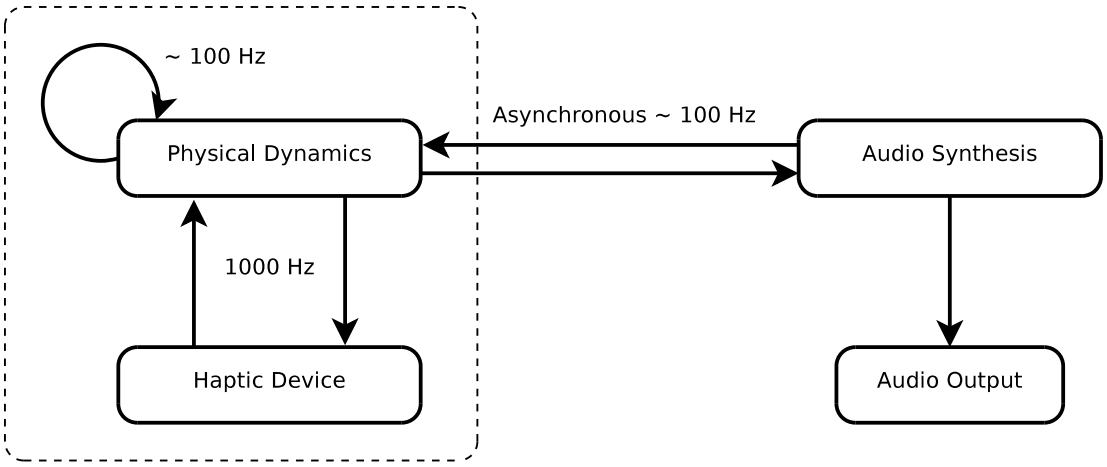


Fig. 4.3 The chosen architecture separates the haptic virtual environment from the sound synthesis over an asynchronous communication channel.

interacts with the environment at 1 KHz. When the haptic proxy touches an object, forces are communicated to the physics thread and applied to the object at the next cycle. This asynchronous approach, based on an example application which is included in the CHAI 3D download, allows objects to feel stiff, since the haptic springs are calculated at an appropriate rate, but also allows the physics model to run slower in order to allow more objects with higher complexity to be modeled.

Events in the physical environment, such as collisions or object movement, are communicated to the audio process using a network protocol. Additionally, the audio process, which is used to control the simulation, can also send messages to the physics model telling it to instantiate or destroy objects, or to change their properties on the fly.

4.5 Communication

In this case, the audio process can be any software which supports the chosen network protocol. For this reason, I decided to exploit the Open Sound Control (OSC) [98] standard for several reasons: it is supported by many popular audio packages that are well-known by the target audience; it allows hierarchical addressing of messages, which is convenient for accessing properties of a scene graph; and it is intended to become a standard for communication between audio systems and gestural controllers.

Open Sound Control messages are composed of an address, which is divided in parts by

the slash character ('/'), a timestamp, and an arbitrary number of arguments which can be any type, such as floating-point numbers, integers, strings, or booleans. For example, in a typical synthesis situation, a controller might change the frequency of an oscillator with a message such as:

```
/oscillator/1/frequency 440
```

For interacting with virtual environments, similar messages are defined for creating and destroying objects, specifying relationships between objects, and modifying and retrieving object properties. This environment can be considered a kind of server for constructing and interacting with virtual musical instruments.

Using OSC it is possible to support standards for querying a controller's parameter namespace and making connections between the controller and the synthesis engine through dedicated mapping tools. We have proposed ideas towards developing such a standard [53], and they will be implemented for this system in the future.

By convention, OSC messages use UDP/IP datagrams as the transport layer, which are considered unreliable but fast, and thus targetted towards data streaming and continuous control. OSC, however, can actually be transported over other types of connections. For example, TCP/IP might be used for more reliable communication over large networks, or a block of shared memory might be used for fast communication on a local computer. For our purposes, UDP/IP was adequate, being quite reliable in practise on a local network, especially since it is usually what is supported by audio software. Informal tests showed that an acceptable number of OSC messages were able to pass between processes on a local network at well under the 24 ms audio-haptic latency mentioned in Section 4.3. OSC was chosen over MIDI, a more established sound protocol, for several reasons: MIDI has limitations on speed, does not support a hierarchical structure, and has no support for floating-point numbers, which are used extensively for data representation in this system.

Some popular audio programs which can send and receive OSC messages are PureData [75], Max/MSP [20], Chuck [95], and SuperCollider [58]. Additionally, almost any of these programs could easily be used to provide a translation layer between a given set of OSC messages and MIDI, opening the possibilities of compatibility even wider.

4.6 Implementation

I have implemented the described system as a C++ software package called DIMPLE, which stands for the Dynamically Interactive Musically PhysicaL Environment. It makes use of several open-source libraries, described in the previous chapter, which allowed me to implement the system quickly and efficiently. Since some of these libraries are licensed under the GNU General Public License, DIMPLE is also GPL software, which means that it is free to download, use, and modify. This choice was made to make the software available to as wide an audience as possible, and to allow others to eventually contribute to development.

DIMPLE makes use of the Open Dynamics Engine (ODE) to control the physical interaction between virtual objects. It uses CHAI 3D to perform high-rate haptic interaction with the objects. The CHAI 3D scene graph is synchronized with ODE at the end of each simulation timestep. It also uses CHAI to update a 3D display, driven by the OpenGL graphics library. A library called LibLo [36] is used for sending and receiving OSC messages. Though CHAI 3D does not support haptic hardware in all operating systems due to availability of hardware drivers, DIMPLE itself is able to run under Microsoft Windows, Linux, and Apple OS X.

A class hierarchy is used to organize objects in the scene and track their CHAI 3D and ODE representations. A diagram of this class organization is given in Figure 4.4. These classes are used for handling OSC message requests. A base class, OscBase, provides mechanisms for registering message handlers. Each type of entity available in DIMPLE has an OSC-enabled class associated with it. These include two types of objects, representing spheres and prisms, as well as six types of constraints. Additionally, composite objects can be constructed which are composed of other objects but share a single instance of ODE's rigid body structure. The constraint types correspond to ODE's "joints", defined in the ODE manual [88]. By creating objects and constraints between them, mechanisms can be created which interact with each other and with the haptic controller.

When a message arrives, it is transferred to the physics and haptics threads for handling. Changes to the CHAI structures occur in the haptics thread, while changes to the ODE structures occur in the physics thread. A common class, OscValue, is used to contain scalar and vector properties of objects, so that any object property can be requested through OSC messaging. These are also kept synchronized with ODE and CHAI properties through the

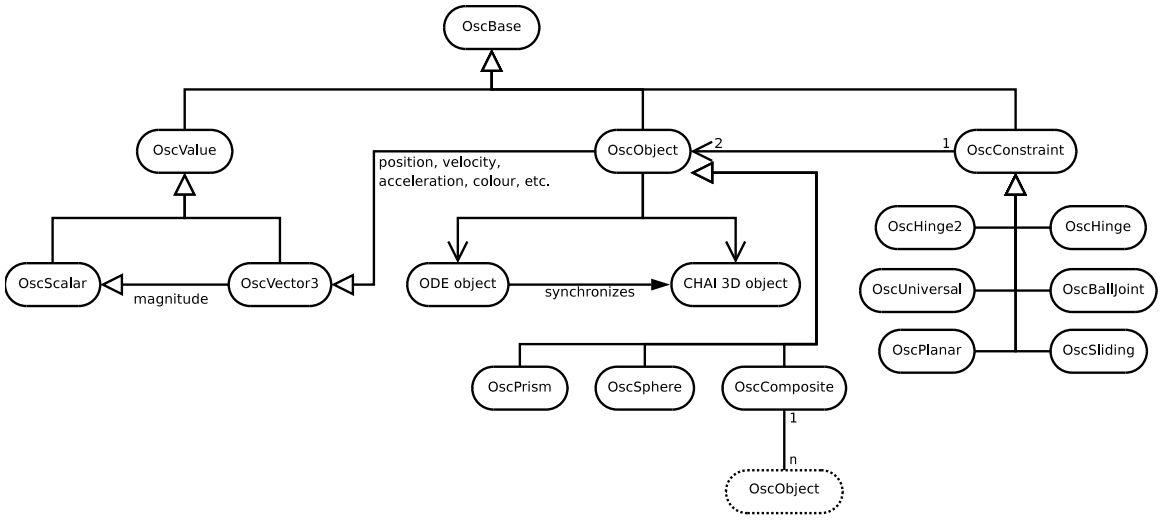


Fig. 4.4 Class diagram of the OSC-aware scene graph. White arrows represent inheritance relationships, open arrows represent pointers, and black arrows represent actions.

use of callback functions.

4.7 Messages

This section includes a brief description of messages accepted by DIMPLE. A complete description can be found in the appendix.

4.7.1 Creating and modifying objects

Objects and constraints are categorized under the major message classes of `/object` and `/constraint`. Objects are created by sending the `create` message addressed to the requested shape or constraint type. Creation messages must be given a name as the first argument. Henceforth the object shall be referred to by name throughout its life. An optional position can also be specified with a 3-vector argument. For example,

```
/object/sphere/create mysphere 0 0.1 0
```

This creates a sphere at the given coordinates. Note that the coordinate system used assumes that the visible screen area in x and y is in the range [-1, 1]. The sphere's radius can then be changed:

```
/object/mysphere/radius 0.5
```

Other properties, described in the appendix, are accessed in a similar manner.

4.7.2 Creating constraints

Constraints can also be created between objects. The keyword `world` is reserved for creating constraints between an object and the "world", referring to the global Cartesian coordinate system. An object constrained by a hinge to the world will seem to be hinged to nothing, free to rotate around a particular axis in space. For example,

```
/constraint/hinge/create hinge1 mysphere world 0 0 0 0 0 1
```

This would hinge the sphere to the point (0, 0, 0), around the axis defined by (0, 0, 1). Note that different constraints require different numbers of points or axes to define them.

4.7.3 Specifying constraint responses

Constraints can also be given *responses*, which are functions defining how the remaining free axes of a constraint should respond to changes in position. An example of a response is a spring. Applying a spring response to a hinge will create an object which pushes back when it is rotated off of its rest position. A response can be specified using a similar message,

```
/constraint/hinge1/response spring 10 0.1
```

This would inform the hinge to respond according to a spring with a stiffness coefficient of 10 and a damping coefficient of 0.1.

4.7.4 Retrieving properties

Properties of objects or constraints can be retrieved by requesting them using the `/get` message. This message takes an optional argument specifying an interval in milliseconds. If this argument is unspecified, the value will be returned to the calling program once. If it is given, the value will be returned at regular intervals. This helps in specifying values to be used as continuous control for modulating parameters of an audio synthesis algorithms. For example,

```
/constraint/hinge1/force/magnitude/get 10
```

This will return the force applied by the constraint's response handler every 10 ms. The value will stop being sent if the same message is sent with a specified interval of zero.

4.8 Virtual Musical Instruments Created

The previous sections introduced a software environment for dynamically creating and experimenting with virtual musical instruments. Here, we describe some simple VMI's that have been created using it. These are not intended to be realistically useful musical instruments, but merely show how the environment can be used. The focus will be on the description of the virtual controller, though audio synthesis will also be discussed briefly. Though most examples are in PureData, similar constructs could certainly be created in other music languages supporting OSC.

4.8.1 Force Stick

As an initial pilot project, I decided to re-implement Verplank's Force Stick [92] in the DIMPLE environment. This seemed like a good starting point, since the force stick is composed of a single bar attached to one actuated joint, which are the minimal components of a DIMPLE instrument.

Initialize the object (named "stick"), and specify its shape, position and mass:

```
/object/prism/create stick
/object/stick/size 0.02 0.02 0.3
/object/stick/position 0 0 0.15
/object/stick/mass 2
```

It can be seen here that once an object is created, it becomes part of the OSC namespace. It can then accept OSC methods which modify it. It is also immediately introduced into the simulation, appears on the screen, and can be touched and manipulated with the haptic device. Next, add a constraint (a hinge) located at the bottom of the prism, named "motor", with a damped spring response:

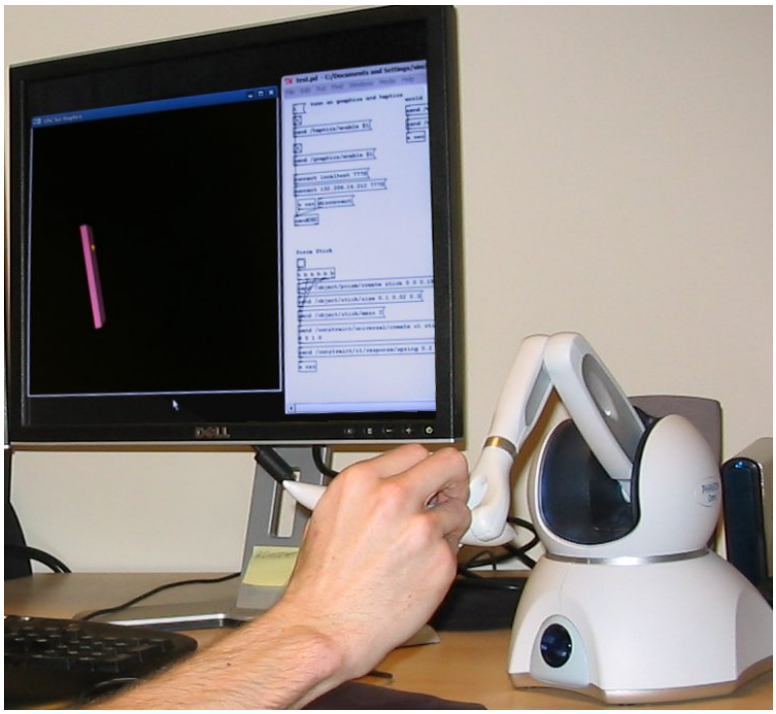


Fig. 4.5 A user playing the virtual Force Stick.

```
/constraint/hinge/create motor stick world 0 0 0 0 1 0
/constraint/motor/response spring 20 1
```

The constraint is located at (0,0,0) and its axis points along (0,1,0), the X-axis. This is the axis around which the stick will rotate.

The constraint is defined to be between the object *stick*, and *world*, indicating a fixed position. The stiffness of the spring action is defined as 20 N·m/rad, and the damping coefficient is 1 N·m·s/rad. The object will not move in space except in rotation around the line segment defined by the given point and axis. The spring, named *motor*, will respond according to the given coefficients.

To change the behaviour of the object when it is pushed or pulled by the device proxy, a different *response* message can be sent to the *motor* constraint. For instance, to get a *squeeze* type of response, as suggested by Verplank, a negative linear response can be used.

```
/constraint/motor/response spring -10
```

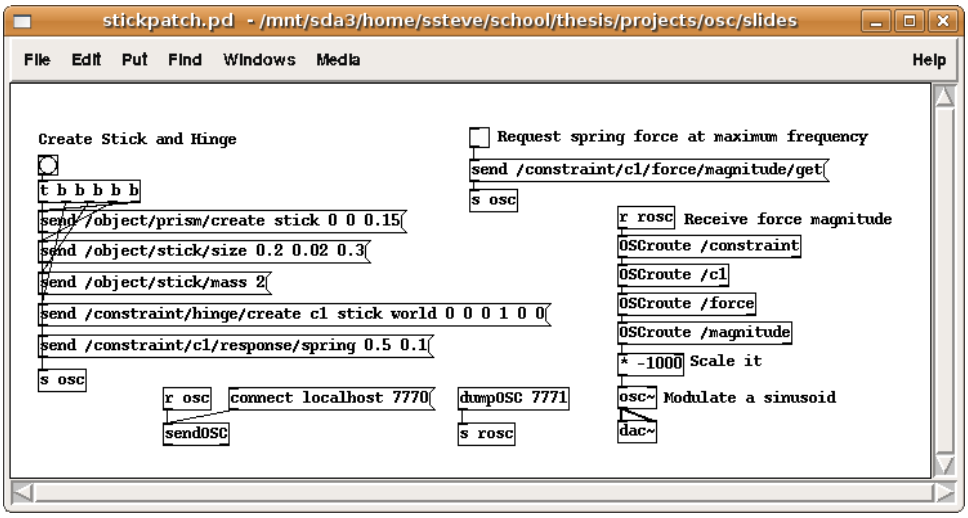


Fig. 4.6 A patch in PureData which creates the Force Stick simulation.

This will reverse the usual spring, so that the stick tends to fall away from the original location, and must be pulled back to the center. "Walls" can be specified on the constraint so that it does not fall all the way around the hinge. No damping was specified here.

To create a sonic response, for example, by modifying the timbre or pitch of a synthesizer, the following message will indicate that the system should send messages every 30 ms:

```
/constraint/motor/force/magnitude/get 30
```

This will cause the force exerted by the stick's constraint to be sent to the audio system at regular intervals. Conversely, the force exerted on the stick itself could be retrieved by,

```
/object/stick/force/magnitude/get 30
```

This simple simulation shows that objects can be created, and user manipulations can produce continuous streams of data that can modulate audio parameters—in this case, the frequency of a sinusoid. A picture of a user manipulating the virtual Force Stick can be seen in Figure 4.5. A patch created in PureData which generates the simulation is shown in Figure 4.6.

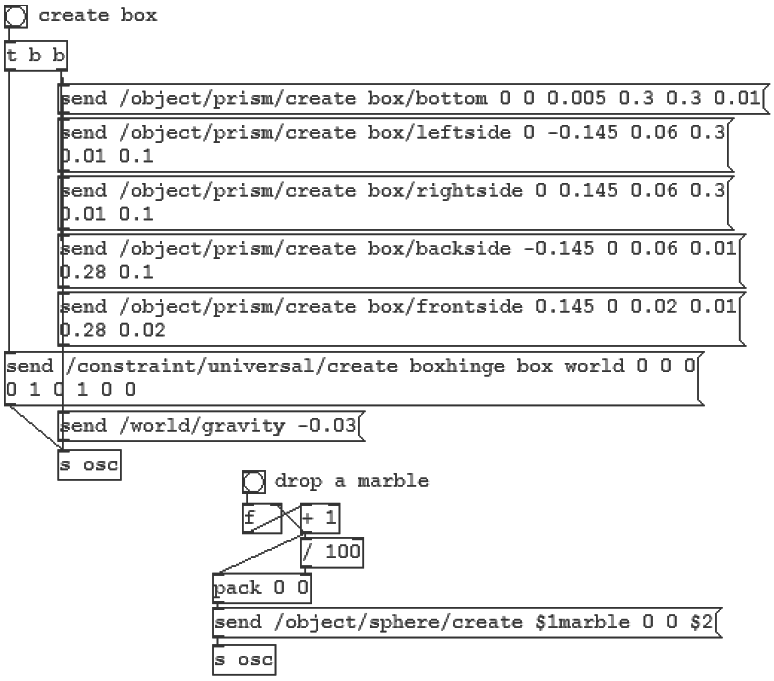


Fig. 4.7 The PureData patch used to generate the MarbleBox simulation. The audio portion of the patch is not shown here. A counting structure is used to generate each new marbles with a unique name.

4.8.2 Marble Box

For a more complex example which might be used to test network latency issues, I decided to simulate the PebbleBox, a controller created by O'Modhrain and Essl [73]. The PebbleBox is a controller using audio analysis to detect collisions between small polished pebbles, which can be used to excite some synthesis engine, such as physical modelling of water or ice cubes. In the virtual version, the pebbles are replaces with spheres which fill a box.

A picture of an implementation in DIMPLE, using only a few messages, can be seen in Figure 4.7 and a screenshot is shown in Figure 4.8. Each part of the box is specified with a `create` message, which is hinged in place so that it is not affected by gravity. Marbles are then dropped into the box. The audio portion receives messages from DIMPLE informing it which objects collided and at what combined velocity. The marbles can be pushed around using the haptic device. Since the CHAI proxy is a point-like sphere which makes interaction with spherical shapes difficult—they role away before they can be pushed—it has been found that it is more interesting to use an ODE body such as a sphere or cube to

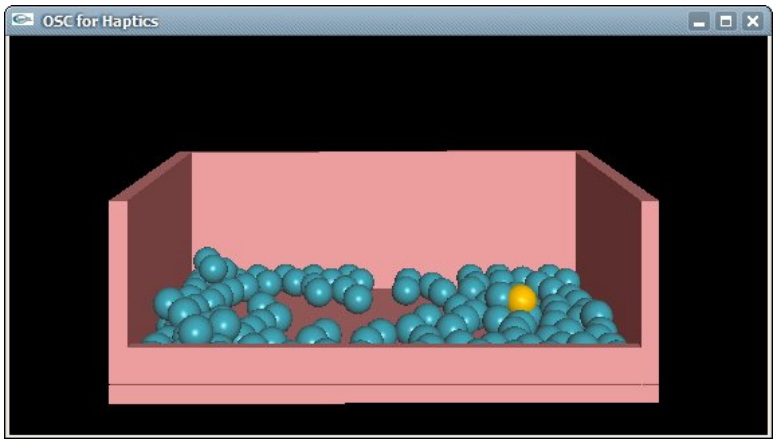


Fig. 4.8 A screenshot of the MarbleBox graphical representation. Here, the light-coloured ball represents the haptic proxy. As marbles hit each other and the sides, they produce triggers for a resonator-bank modal synthesis algorithm.

push around the marbles by "grabbing" it with the haptic proxy using a stiff spring. This functionality has been encapsulated in DIMPLE by specifying the message,

`/object/<name>/grab`

When an object is grabbed, it follows the movement of the haptic end effector, but it conversely allows the haptic device to give an impression of the object's weight and the feeling of bumping into other objects in the scene.

A simple audio model was used, consisting of a decaying envelope applied to a low-frequency sinusoid, so that the spheres seemed to make a small "bumping" sound. However, in a later experiment, these collision messages were forwarded to a proper modal synthesis algorithm, running in Max/MSP on a separate computer. This seemed to give the marbles a perceptually metallic quality, and showed how easy it was to use OSC for easily linking to another synthesis engine—the PureData model was running on Microsoft Windows, while Max/MSP was running on a Mac on the local network. Perceptually, latency did not seem to be a problem.

For comparison, two previous virtual implementations of the PebbleBox have been created in the Enactive Network [50]. One of these simulations used the same physical dynamics engine (ODE) as the implementation discussed here, while the other used CORDIS-ANIMA. The ODE simulation was written in C++ and used Microsoft Direct3DSound for

performing spatialization on playback of soundfiles. Spatialization was controlled by the positions of the marbles. Similar to this simulation, the marbles could be pushed around with the haptic device. Haptics were implemented using OpenHaptics from SensAble. The CORDIS-ANIMA implementation was two-dimensional. It consisted of a large circle containing several smaller circles. The movement parts were executed at 3 KHz, while the sounding parts of the model were calculated at 30 KHz. It used an ERGOS device interfacing with the simulation at 3 KHz. In contrast the ODE/Phantom simulation ran at 1 KHz. It was reported that, "the ERGOS haptic device was quoted as providing a more distinct perception of the local surface properties of each of the objects," though it is not clear what aspect of the device, such as update rate, stiffness, or ergonomics, was most responsible. The report discusses an interesting aspect of this simulation which is that it contains the presence of both direct audio feedback related to movement as well as indirect feedback related to objects colliding against each other. Users found direct feedback "easier to understand".

4.8.3 Chained FM

Since the MarbleBox was oriented around event-style data, an attempt was created to provide continuous control for FM synthesis. Additionally, I was interested to see how objects would behave when connected through hinges in a serial fashion. In Chained FM, each successive prism is connected to the prism to its left by a hinge. The left-most prism is hinged to the world. Pushing against a prism causes a chain reaction in which the other prisms follow suit. Since each hinge is given a spring response, movement leads to coupled oscillating behaviour. In the case shown here, the velocity of the objects is used to modulate the carrier and modulator frequencies of two FM pairs. In practise, velocity may not be the most interesting parameter for this purpose, but it was sufficient to show how multi-parametric changes can be modified in a related way by coupling objects in the scene. A picture of the interface is given in Figure 4.9 and the PureData patch is shown in Figure 4.10.

4.8.4 Rolling Balls and Cannon Balls

In collaboration with Mark Mashall and Joe Malloch, who are also students in McGill's Input Devices and Music Interaction Laboratory, we explored the use of DIMPLE for

Fig. 4.9 A photograph of a user interacting with Chained FM using a SensAble Phantom Desktop. The prism objects are connected by springs. The movement of these objects produces parametric changes in an FM synthesis algorithm.

non-haptic tasks. In one instance, we directed the gravity vector of the MarbleBox VMI according to the center of balance determined by a force-sensing floor, causing the marbles to roll to one side or another in correspondence with the user's posture. The ball positions were then used to control a spatialization algorithm.

In another set-up, we connected several drum pads from a MIDI drum set to Max/MSP. We caused balls to be created when a drum was hit, and sent flying in a direction according to the position of the drum relative to the player. Again the moving position of each created ball was used to control spatialization. These two experiments were done for a project related to the gestural control of spatialization [56].

The advantage of using OSC here was clear: since controllers for the force floor and drum pads had already be created in Max/MSP, we were able to very quickly connect previously made modules in no more than a few minutes to create new demonstrations that took advantage of DIMPLE's physical dynamics engine.

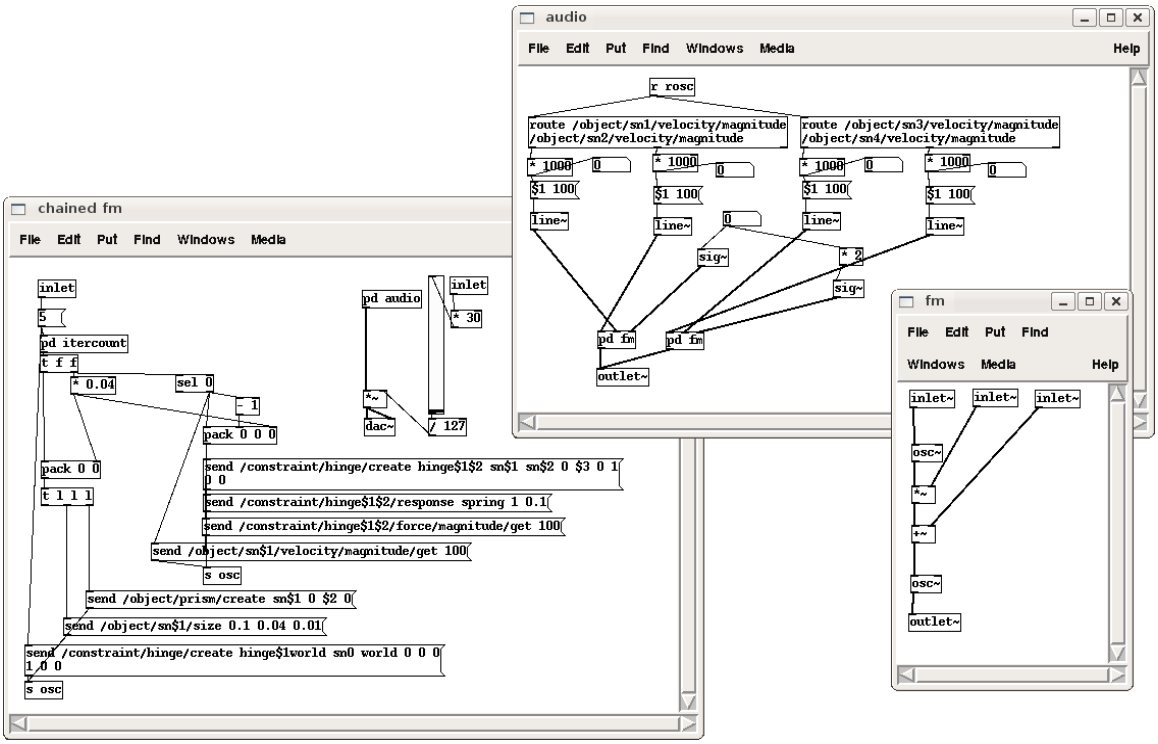


Fig. 4.10 The PureData patch used to generate Chained FM. The left-most window shows the creation of the prism objects and their hinges. The middle patch shows the receiving portion, in which parameters are sent to the right-most patch, which contains the FM synthesis pairs. Here, velocity is used to control modulator and carrier frequencies.

4.9 Summary

A system has been presented for running a haptically-enabled virtual environment in a process which can asynchronously communicate with several existing audio software packages using an increasingly well-supported communication protocol. Objects can be created as well as several types of constraints on these objects. A haptic device can be used to interact with these objects, allowing the sensation of touch. Properties of objects can be requested using the OSC protocol, and used for modulating audio synthesis parameters.

We have shown the use of the system for creating a few simple virtual musical instruments that can be touched. There are, however, several other constraint types to explore, and neither constraints between objects, nor the use of various constraint responses has been fully exploited. Future work will certainly reveal the use of these features. The next

chapter proposes some expansions to the system, and describe tentative experimental work which may take advantage of DIMPLE's dynamic nature and ease of use.

Chapter 5

Conclusions and Future Work

This chapter will present some conclusions drawn from the experience of creating this system, and provide an outline of some work to be carried out in the future which will make use of its unique features and also suggest the addition of new functionalities.

5.1 Conclusions and Applications

In this thesis, I have described the system I developed for creating and interacting with physically active virtual musical instruments using haptic force-feedback technology.

While the features offered by this software do not necessarily allow the simulation of all possible virtual instruments, it is hoped that a good cross-section of basic interaction paradigms can be created, as well as a few that would be problematic with real objects. Hitting gestures, plucking, pushing, and pulling are all possible. With some basic extensions described below in Section 5.2.3, it may be possible to take advantage of rubbing, scrubbing, and bowing gestures. Additionally, dynamically modifying physical parameters such as the gravity vector, object mass, and stiffness, removing and adding objects in the scene, and creating and breaking constraints between objects can allow simulation of interactions that might be quite difficult to accomplish using sensors and mechanical systems.

Haptics can provide a more immersive virtual experience. Feedback through the haptic channel can be a rich source of information about the interface and algorithm being manipulated. For virtual instruments, where the control surface is otherwise not tangible, providing a haptic sense of the environment can help a performer to maintain awareness of the metaphor that is being used to control sonic output. Multi-modal displays have applica-

tion in psychology and music technology research, as well as for haptic device evaluation in the commercial market; if it can be shown that there are certain minimal performance measures for satisfying musical control over a given virtual instrument, such as those described by Hayward and Astley [38], then new devices can be evaluated accordingly. The performance of a haptic device may be critical to applications such as virtual surgery, and finding correlations with musical interaction may be useful for manufacturers and researchers who need access to moderately large numbers of subjects who are skilled in gestural control.

I hope that by providing this software to the music and research communities, it will help to promote the use of haptic technology for enhancing and investigating musical interaction. While it is important to keep the limitations of pen-based haptic display in mind, virtualized musical interfaces provide a means of making available a practically unlimited set of gestural control possibilities without requiring the cost and difficulty of custom hardware development. With haptic devices becoming available at lower cost, it is an exciting time to be part of this growing research community. Making haptic technology and physically dynamic virtual environments available to a large subset of music and audio software should provide a convenient means for musicians and researchers alike to explore the possibilities offered by it.

5.2 Ideas for future work

This section will present some ideas for expansion of the system, and ruminations on future research directions.

5.2.1 Object shapes

The given system can be used to create virtual environments which make use of a limited set of 3D primitives, and several available constraints can be applied to their relative or absolute movement. While more complicated compound objects can be created by "gluing" together these primitives, not all possible or even desirable shapes and interactions can be described in this way. Adding more primitives to the environment is one way to expand the tool box. However, the support of mesh-based objects would also be quite useful. It would allow manipulation of objects resembling structures in real life or in the imagination, which could be designed using capable external tools. Since DIMPLE makes use of CHAI 3D, it would make sense to allow it to load 3DS files, which is a functionality it already supports.

ODE does support physical dynamics involving mesh objects, though it is necessarily slower than primitive-based collision detection. It may be possible to incorporate more complex algorithms for fast collision handling in higher polygon environments if necessary. Prof. Ming C. Lin of the University of North Carolina at Chapel Hill, for example, has a large body of research dedicated to optimizing mesh-based collision detection [27, 35, 47, 97].

5.2.2 Deformable objects

Objects in the environment are currently limited to rigid bodies, but the manipulation of virtual "clay" or other deformable models would provide an interesting avenue for musical control. Mulder found informally that deforming virtual objects in ways that seemed intuitively related to certain sound parameters was beneficial to performance.

Many haptic software packages, such as OpenHaptics or the Reachin API, come with a demonstration of a deformable membrane. This is sometimes intended to simulate the idea of poking a patient's skin with a needle in a medical simulation. However, the idea of injecting energy into a deformable system to cause oscillations is used by several musical paradigms: the CORDIS-ANIMA and Cymatic systems use it at the level of audio frequency oscillations to create simulations of vibrating bodies, while Verplank's scanned synthesis uses large oscillations of a string model to control a related audio algorithm. Thus, it would seem interesting to be able to deform virtual objects within this system. Maintaining physical realism and supporting various types of deformation can be computationally demanding, but it remains nonetheless an interesting avenue for exploration. OSC could likely be used to transmit the positions of an object's vertex points without imposing too much overhead.

5.2.3 Texture and friction

Though her results were surprising, O'Modhrain [71] showed that friction certainly has an effect on the playability of a haptic instrument. Richard and Cutkosky [78] reviewed several friction models for haptics, also proposing a new one. Friction is a source of vibrotactile feedback which is not desirable to remove from force feedback simulations, since virtual objects would otherwise feel "slippery." Fortunately, CHAI 3D is already equipped to handle multiple friction models, but currently only provides one. Since friction has a particular effect in musical interactions—namely, to inject a certain pattern of energy into

a vibrating system—it would be interesting to add support for several friction models, and also to determine how best the information about micro-interactions could be summarized and transmitted to the audio engine. This problem has been previously investigated by van den Doel et al. [90], and incorporating and improving their findings would be interesting.

Additionally, as seen in the Reachin API [77], it is possible to give objects a haptic texture using a bump map. These textured micro-interactions would certainly have applicability in musical interaction. Fine-grained textures would make an interesting study case for examining the importance of a haptic device's physical precision.

5.2.4 Vibrotactile feedback

Related to this idea, in terms of the CORDIS-ANIMA model especially, is the idea that audio frequency vibrations contain an important portion of the energy injected into the system by user interaction. This audio vibration, which is what supposedly makes an instrument feel "warm" and "alive", and which Chafe [15] and others have shown to be detectable by performers, helps give a user feedback about the relationship between control changes and sound output. Though this project has decidedly concentrated on force feedback over vibrotactile feedback, ignoring this rich source of information would be a mistake. Currently there exists software solutions allowing processes on a computer or network to transmit audio streams to each other with low latency. One example is the JACK system [23], designed on Linux but also available for Apple OS X. Providing the ability to route this information into the control surface or even directly into the haptic controller may be an interesting possibility.

5.2.5 Experimental research

Creating virtual musical instruments can allow the exploration of aspects of haptic interaction that would be impossible with real instruments. Experimental research depends almost entirely on separating variables so that they can be explored in a methodical fashion. In an acoustic instrument, it is impossible or very difficult to separate the vibrations of a body from the shape, weight, and texture of it. Even in an electronic gestural controller, it can be problematic to remove the texture of the surface, the feel of buttons or often the outline of many types of sensors. When an instrument is completely virtualized, it can be played with or without being felt, and the continuum between having rich tactual information and

having none at all can be explored. Haptic variables such as weight and stiffness can be modified independantly.

Within the Enactive Network, cross-modal studies have been proposed such as target-finding tasks in which haptic and sonic information is modulated in various ways according to distance. In another discussion, it was suggested to explore the addition of auditory stimuli to a haptic experiment involving simulation of a virtual rolling ball [99]. Subjects were able to judge the length of a virtual tube more easily when continuous haptic feedback was present, in comparison to only the difference in time between the beginning and end of the roll. Auditory feedback would provide more permutations to the experiment. Within my own work, I would like to examine how the presence of inertia and haptic contour of a control surface affects the playability of a musical interface, and also how the presence of visual stimulus affects the results.

Appendix A

OSC messages implemented in DIMPLE

This document specifies messages which are handled or broadcast by DIMPLE.

A.1 Global parameters

These parameters are global to the whole simulated world.

- `/haptics/enable` *boolean*
  Enable the haptic device. Prior to enabling haptics, only the physics thread is running. If *boolean* is zero, haptics will be disabled.

- `/graphics/enable` *boolean*
  Enable the graptic window. Prior to enabling graphics, the physics thread is running with no visual display. If *boolean* is zero, the graphics window will close.

- `/world/gravity` x y z
  `/world/gravity` z
  Specify the world gravity vector. If only one parameter is specified, it is assumed to be the magnitude of a downward-pointing vector.

- `/world/clear`
  Clear all objects in the world.

A.2 Objects

Objects are the physical objects which move in the environment, collide with each other, and can be touched using the haptic proxy.

A.2.1 Object creation

These messages are for creating objects of simple geometric primitives.

- `/object/sphere/create` *name x y z*
  Create a sphere named *name* at the given position in space, with a default radius of 0.01.

- `/object/prism/create` *name x y z*
  Create a prism named *name* at the given position in space, with a default size of (0.01, 0.01, 0.01).

A.2.2 Object methods

These messages make some modification to an object.

- `/object/`*name*`/destroy`
  Destroy the object named *name*.

- `/object/`*name*`/grab`
  "Grabbing" an object means introducing a stiff two-way spring between the position of the object and the haptic proxy object. In effect, movement of the device causes the object to following quickly, and movement of the object causes pulling forces on the end effector. The user thus feels the object's weight, and the illusion of the object being associated with the haptic device is created.

- `/object/`*name*`/ungrab`
  If an object is grabbed, this method lets it go.

A.2.3 Object attributes

These messages represent an object's attributes. They can be set by sending the message to DIMPLE, or requested as described in Section A.4, below.

Object size

- `/object/`*name*`/radius` *radius*
  `/object/`*name*`/size` *width height depth*
  Specify the radius of a sphere or the size of a prism. The mass will be scaled correspondingly.

Object materials

- `/object/`*name*`/mass` *mass*
  The object's weight in grams.
- `/object/`*name*`/color` *r g b*
  The object's colour in RGB values between 0 and 1.
- `/object/`*name*`/friction` *static dynamic*
  The static and dynamic friction coefficients for *name*.
- `/object/`*name*`/stiffness` *stiffness*
  The object's stiffness coefficient.

Object movement

- `/object/`*name*`/position/magnitude` *magnitude* [1]
  `/object/`*name*`/position` *x y z*
  The object's position. Setting this attribute will "beam" it to the new position immediately.
- `/object/`*name*`/velocity/magnitude` *magnitude*
  `/object/`*name*`/velocity` *x y z*
  The object's velocity. Setting this attribute will cause unnatural movement of the object.
- `/object/`*name*`/acceleration/magnitude` *magnitude*
  `/object/`*name*`/acceleration` *x y z*

[1] Setting a vector's magnitude will work only if the current magnitude is non-zero.

The object's acceleration. Setting this attribute is equivalent to applying a force without regard for the object's mass.

- `/object/`*name*`/force` *x y z*
  Apply the given force vector to *name*, or retrieve the force applied to an object through collisions or interaction with the proxy object.

A.3 Constraints

A constraint consists of some way in which two objects, or one object and a fixed position, are related. Often this is a case where two objects are joined by some kind of joint, and the movement on said joint can be restricted or given some kind of response behaviour.

A.3.1 Constraint creation

Each constraint type requires a different set of attributes, by definition. The reader is refered to the Open Dynamics Engine manual [88] for more information on these constraint types and the attributes they require.

- `/constraint/fixed/create` *name object object* $point_x$ $point_y$ $point_z$
- `/constraint/ball/create` *name object object* $point_x$ $point_y$ $point_z$
- `/constraint/hinge/create` *name object object* $point_x$ $point_y$ $point_z$ $axis_x$ $axis_y$ $axis_z$
- `/constraint/hinge2/create` *name object object* $point_x$ $point_y$ $point_z$ $axis1_x$ $axis1_y$ $axis1_z$ $axis2_x$ $axis2_y$ $axis2_z$
- `/constraint/sliding/create` *name object object* $point_x$ $point_y$ $point_z$ $axis_x$ $axis_y$ $axis_z$
- `/constraint/universal/create` *name object object* $point_x$ $point_y$ $point_z$ $axis1_x$ $axis1_y$ $axis1_z$ $axis2_x$ $axis2_y$ $axis2_z$

In the above, *object* may be `world`, to indicate a constraint against a fixed position.

A.3.2 Constraint methods

- `/constraint/`*name*`/destroy`
  Destroy the constraint named *name*. Note that constraints will also be destroyed if one of the associated objects is destroyed.

A.3.3 Constraint responses

Each constraint can be given a response characteristic, which specifies how it should behave as the user pushes the constraint away from the fulcrum of the constraint.

- `/constraint/`*name*`/response` *response ...*
  Where *response* is one of:
  - `spring` *stiffness damping*
    Response is determined by a damped spring equation. If *damping* is not specified, an undamped spring is used.
  - `constant` *value*
    Response presents a constant force against the direction of movement. Feels similar to moving a stiff hinge. Supplying zero for *value* will remove any response so that the constraint moves freely.
  - `noise` *threshold*
    Response presents a noisy graininess in movement, like moving against sand paper.
  - `pluck` *position stiffness*
    Multiple `pluck` response messages may be accumulated at different positions. It corresponds to a "membrane" which gives some resistance before breaking through. *position* here is an angle in radians. *stiffness* is optional, and has a reasonable default.
  - `wall` *position direction*
    Walls delimit the range of motion for a constraint. In other words, any motion past the given *position*, in the given *direction*, be treated like an infinitely stiff spring. (In practise, a "very" stiff spring will be used.) *position* is an angle in radians, and *direction* is either 1 or -1.

A.3.4 Constraint attributes

Constraint attributes can be set by sending the message to DIMPLE, or requested as described in Section A.4, below.

- `/constraint/`*name*`/force/magnitude` *magnitude*
  `/constraint/`*name*`/force` *x y z*
  The current force acting on the constraint.

A.4 Requesting information

A.4.1 Requesting attributes

Any attributes of objects and constraints listed in the previous section may be of interest to the audio and control systems. Thus any of the attributes may take the `/get` method to retrieve corresponding value.

- `.../`*attribute*`/get` *interval*
  Specify *interval* = 0 to stop. *interval* may be omitted to get the attribute only once.

The `get` method may include an optional interval in milliseconds which will tell the haptic system to report the corresponding attribute continuously at regular intervals.

The returned attribute will have the same OSC message address, but without `/get`.

A.4.2 Requesting collisions

The following messages can be used to request collision information. Collisions currently provide only force information. (The repelling force magnitude required for a perfectly elastic collision.)

- `/object/collide/get`
  `/object/collide/get` *boolean*
  Request that all inter-object collisions be reported. If *boolean* is not provided, it is assumed to be 1. If *boolean* = 0, collisions will no longer be reported.
  Collisions are reported with the message,
  `/object/collide` *object object force*

- `/object/`*name*`/collide/get`
  `/object/`*name*`/collide/get` *boolean*
  Request that all collisions with object *name* are reported. If *boolean* is not provided, it is assumed to be 1. If *boolean* = 0, collisions will no longer be reported for this object.
  Collisions are reported with the message,
  `/object/`*name*`/collide` *force*

References

[1] M. Adcock, M. Hutchins, and C. Gunn. Augmented reality haptics: Using ARToolKit for display of haptic applications. In *Proceedings of the 2nd International Augmented Reality Toolkit Workshop*, pages 1–2, Tokyo, Japan, October 2003.

[2] B. D. Adelstein, D. R. Begault, M. R. Anderson, and E. M. Wenzel:. Sensitivity to haptic-audio asynchrony. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, pages 73–76, Vancouver, BC, November 2003. ACM.

[3] A. Backman and D. Sjölie. osgHaptics (software), 2007.
Available: `http://www.vrlab.umu.se/research/osgHaptics`.

[4] R. Bargar, I. Choi, S. Das, and C. Goudeseune. Model-based interactive sound for an immersive virtual environment. In *Proceedings of the International Computer Music Conference*. ICMA, 1994.

[5] W. V. Baxter III and M. C. Lin. A versatile interactive 3D brush model. In *Proceedings of the Pacific Conference on Computer Graphics and Applications*, pages 319–328, 2004.

[6] P. D. Bennett. Hap-Kit: A haptic interface for a virtual drum-kit. In *The Symposium for Cybernetics Annual Research Projects*, University of Reading, June 2004.

[7] P. D. Bennett, N. Ward, S. O'Modhrain, and P. Rebelo. DAMPER: a platform for effortful interface development. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 273–276, New York, NY, June 2007.

[8] D. Birnbaum. The Touch Flute: Exploring roles of vibrotactile feedback in musical performance. Project report, McGill University, 2003.

[9] B. Bongers. Tactual display of sound properties in electronic musical instruments. *Displays*, 18:129–133, 1998.

[10] G. C. Burdea. *Force and touch feedback for virtual reality*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[11] C. Cadoz. Instrumental gesture and musical composition. In *Proceedings of the 14th International Computer Music Conference*, pages 1–12, Köln, 1988. ICMA.

[12] C. Cadoz and M. M. Wanderley. Gesture-music. In M. Wanderley and M. Battier, editors, *Trends in Gestural Control of Music*, pages 71–94. IRCAM, Paris, France, 2000.

[13] C. Cadoz, L. Lisowski, and J.-L. Florens. A modular feedback keyboard design. *Computer Music Journal*, 8(1):48–51, 1990.

[14] C. Cadoz, A. Luciani, and J. L. Florens. CORDIS-ANIMA: a modeling and simulation system for sound and image synthesis—the general formalism. *Computer Music Journal*, 17(1):19–29, 1993. MIT Press.

[15] C. Chafe. Tactile audio feedback. In *Proceedings of the International Computer Music Conference*, pages 76–79. ICMA, 1993.

[16] C. Chafe and S. O'Modhrain. Musical muscle memory and the haptic display of performance nuance. In *Proceedings of the International Computer Music Conference*, pages 428–431, Hong Kong, 1996. ICMA.

[17] I. Choi, R. Bargar, and C. Goudeseune. A manifold interface for a high dimensional control interface. In *Proceedings of the International Computer Music Conference*, pages 385–392, San Francisco, 1995. International Computer Music Association, ICMA.

[18] F. Conti, , F. Barbagli, R. Balaniuk, M. Halg, C. Lu, D. M. L. Sentis, E. Vileshin, J. Warren, O. Khatib, and K. Salisbury. The CHAI Libraries. In *Eurohaptics '03*, pages 496–500, Dublin, Ireland, June 2003.

[19] J. M. Couturier. A model for graphical interaction applied to gestural control of sound. In *Proceedings of the international conference on Sound and Music Computing*, Marseille, May 2006.

[20] Cycling '74. Max/MSP (software), 2007. Available: `http://www.cycling74.com`.

[21] M. Danks. Real-time image and video processing in GEM. In *Proceedings of the International Computer Music Conference*, pages 220–223. ICMA, 1997.

[22] P. L. Davidson and J. Y. Han. Synthesis and control on large scale multi-touch sensing displays. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 216–219, Paris, France, 2006. IRCAM—Centre Pompidou.

[23] Davis, P. et al. JACK Audio Connection Kit (software).
Available: `http://jackaudio.org`.

[24] M. de Pascale and D. Prattichizzo. Haptik Library: A component based architecture for uniform access to haptic devices. To appear in IEEE Robotics & Automation Magazine, 2007.

[25] T. DeFanti, P. Banerjee, C. Luciano, and S. Mehrotra. Realistic cross-platform haptic applications using freely-available libraries. In *Proceedings of the 12th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, in conjunction with IEEE Virtual Reality*, pages 282– 289, Los Alamitos, CA, April 2004.

[26] D. DiFilippo and D. K. Pai. The AHI: An audio and haptic interface for contact interactions. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 149–158, 2000.

[27] S. A. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20, pages 500–510. Blackwell Publishing, 2001.

[28] ERGOS Technologies. ERGOS haptic devices (hardware).
Available: `http://www-acroe.imag.fr/ergos-technologies`.

[29] J. Feasel. A virtual stringed instrument with haptic feedback.
Available: `http://www.cs.unc.edu/~feasel/classes/259/final/results.html`, 2003.

[30] J.-L. Florens. Expressive bowing on a virtual string instrument. In A. Camurri and G. Volpe, editors, *Lecture Notes in Computer Science: Gesture-Based Communication in Human-Computer Interaction, 5th International Gesture Workshop*, volume 2915, pages 487–496. Springer Berlin / Heidelberg, Genova, Italy, April 2003.

[31] J.-L. Florens, A. Luciani, and N. Castagné. Sharp colliding of multiple sound objects with haptic feedback. Enactive Network, 2005.

[32] Force Dimension. Omega & Delta haptic devices (hardware).
Available: `http://www.forcedimension.com`.

[33] B. Gillespie. *Haptic Display of Systems with Changing Kinematic Constraints: The Virtual Piano Action*. PhD thesis, Stanford University, January 1996.

[34] R. B. Gillespie and S. O'Modhrain. The Moose: A haptic user interface for blind persons with application to the digital sound studio. Technical Report STAN-M-95, Stanford University Department of Music, October 1995.

[35] N. K. Govindaraju, S. Redon, M. C. Lin, and D. Manocha. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In

HWWS '03: Proceedings of the ACM SIGGRAPH/Eurographics conference on graphics hardware, pages 25–32, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[36] S. Harris and N. Humfrey. LibLo: Lightweight OSC implementation. Available: `http://liblo.sourceforge.net/`, January 2007.

[37] V. Hayward and B. Armstrong. A new computational model of friction applied to haptic rendering. In *Proceedings of The Sixth International Symposium on Experimental Robotics VI*, pages 403–412, London, UK, March 2000. Springer Verlag.

[38] V. Hayward and O. R. Astley. Performance measures for haptic interfaces. In G. Giralt and G. Hirzinger, editors, *Robotics Research: The 7th International Symposium*, pages 195–207, Berlin, Germany, 1996. Springer Verlag.

[39] V. Hayward, O. R. Astley, M. Cruz-Hernandez, D. Grant, and G. Robles-De-La-Torre. Haptic interfaces and devices. *Sensor Review*, 24(1):16–29, 2004.

[40] T. Hermann, J. Krause, and H. Ritter. Real-time control of sonification models with a haptic interface. In *Proceedings of the International Conference on Auditory Display*, pages 82–86, Kyoto, Japan, July 2002. International Community for Auditory Display.

[41] D. Howard, S. Rimell, A. Hunt, P. Kirk, and A. Tyrrell. Tactile feedback in the control of a physical modelling music synthesiser. In *Proceedings of the 7th International Conference on Music Perception and Cognition*, pages 224–227, 2002.

[42] A. Hunt, M. Wanderley, and M. Paradis. The importance of parameter mapping in electronic instrument design. In *Proceedings of the 2002 Conference on New Interfaces for Musical Expression*, pages 149–154, 2002.

[43] R. J. K. Jacob, L. E. Sibert, D. C. McFarlane, and J. M. Preston Mullen. Integrality and separability of input devices. *ACM Transactions on Computer-Human Interaction*, 1(1):3–26, 1994.

[44] S. Jordà. Sonigraphical instruments: from FMOL to the reacTable. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 70–76, Montreal, Canada, 2003.

[45] M. Karjalainen and T. Mäki-Patola. Physics-based modeling of musical instruments for interactive virtual reality. In *Proceedings of the International Workshop on Multimedia Signal Processing*, pages 223–226, Siena, Italy, September 2004.

[46] S. D. Laycock and A. M. Day. Recent developments and applications of haptic devices. *Computer Graphics Forum*, 22(2):117–132, June 2003.

[47] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *Proceedings of the IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[48] A. Luciani, J.-L. Florens, and N. Castagné. From action to sound: a challenging perspective for haptics. In *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC'05)*, pages 592–595, Pisa, Italy, March 2005. IEEE Computer Society.

[49] N. Magnenat-Thalmann, M. Montagnol, R. Gupta, and P. Volino. Interactive virtual hair-dressing room. In *CAD Conference 2006, International Journal on Computer-Aided Design & Applications*, pages 535–545, Phuket Island, Thailand, June 2006.

[50] C. Magnusson, A. Luciani, D. Couroussé, R. Davies, and J.-L. Florens. Preliminary test in a complex virtual dynamic haptic audio environment. In *2nd Enactive Workshop*, McGill University, Canada, May 2006.

[51] T. Magnusson. ixi software: The interface as instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 212–215, Vancouver, Canada, May 2005.

[52] T. Mäki-Patola and P. Hämäläinen. Latency tolerance for gesture controlled continuous sound instrument without tactile feedback. In *Proceedings of the International Computer Music Conference*, pages 11–16, Miami, USA, November 2004. ICMA.

[53] J. Malloch, S. Sinclair, and M. M. Wanderley. From controller to sound: Tools for collaborative development of digital musical instruments. In *Proceedings of the International Computer Music Conference*, Copenhagen, Denmark, August 2007. ICMA. Accepted, in press.

[54] W. R. Mark, S. C. Randolph, M. Finch, J. M. V. Verth, and I. Russell M. Taylor. Adding force feedback to graphics systems: issues and solutions. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 447–452, New York, NY, USA, 1996. ACM Press.

[55] M. T. Marshall and M. M. Wanderley. Vibrotactile feedback in digital musical instruments. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 226–229, Paris, France, June 2006. IRCAM—Centre Pompidou.

[56] M. T. Marshall, N. Peters, A. R. Jensenius, and J. Boissinot. On the development of a system for gesture control of spatialization. In *Proceedings of the International Computer Music Conference*, pages 360–266, New Orleans, LA, November 2006. ICMA.

[57] T. Massie. A tangible goal for 3D modeling. *Computer Graphics and Applications, IEEE*, 18(3):62–65, May/June 1998.

[58] J. McCartney. Rethinking the computer music language: SuperCollider. *Computer Music Journal*, 26:61–68, 2002.

[59] M. Minsky, O.-Y. Ming, O. Steele, F. P. Brooks Jr., and M. Behensky. Feeling and seeing: issues in force display. In *SI3D '90: Proceedings of the Symposium on Interactive 3D Graphics*, pages 235–241, New York, NY, USA, 1990. ACM Press.

[60] MPB Technologies Inc. Freedom 6S & 7S haptic devices (hardware). Available: `http://www.mpb-technologies.ca`.

[61] A. Mulder. Virtual musical instruments: Accessing the sound synthesis universe as a performer. In *Proceedings of the First Brazilian Symposium on Computer Music*, pages 243–250, 1994.

[62] A. Mulder. *Design of Virtual Three-dimensional Instruments for Sound Control*. PhD thesis, Simon Fraser University, 1998.

[63] A. Mulder. Towards a choice of gestural constraints for instrumental performers. In M. Wanderley and M. Battier, editors, *Trends in Gestural Control of Music*, pages 321–352. IRCAM, Paris, France, 2000.

[64] A. G. Mulder, S. S. Fels, and K. Mase. Empty-handed gesture analysis in Max/FTS. In A. Camurri, editor, *Proceedings of the AIMI international workshop on Kansei—the technology of emotion*, pages 87–90, Genova, Italy, October 1997.

[65] G. K. Ng. A virtual environment for instrumental music performance. Master's thesis, Faculty of Science, University of Manchester, Manchester, UK, November 1994.

[66] C. Nichols. The vBow: development of a virtual violin bow haptic human-computer interface. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 29–32, Dublin, Ireland, 2002.

[67] Novint Technologies Inc. Falcon haptic devices (hardware). Available: `http://www.forcedimension.com/fd/avs/home/products/`.

[68] I. Oakley, M. R. McGee, S. Brewster, and P. Gray. Putting the feel in 'look and feel'. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 415–422, The Hague, The Netherlands, 2000.

[69] R. Oboe and G. D. Poli. Multi-instrument virtual keyboard: the MIKEY project. In *Proceedings of the 2002 Conference on New Interfaces for Musical Expression*, pages 23–28, Dublin, Ireland, May 2002. Media Lab Europe.

[70] J. F. O'Brien, C. Shen, and C. M. Gatchalian. Synthesizing sounds from rigid-body simulations. In *The ACM SIGGRAPH 2002 Symposium on Computer Animation*, pages 175–181. ACM Press, July 2002.

[71] S. M. O'Modhrain. *Playing by Feel: Incorporating Haptic Feedback into Computer-Based musical Instruments*. PhD thesis, Stanford University, November 2000.

[72] S. M. O'Modhrain and C. Chafe. Incorporating haptic feedback into interfaces for music applications. In *Proceedings of the International Symposium on Robotics with Applications, World Automation Conference*, 2000.

[73] S. M. O'Modhrain and G. Essl. PebbleBox and CrumbleBag: Tactile interfaces for granular synthesis. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 74–79, Hamamatsu, Japan, August 2004.

[74] Osfield, R. et al. OpenSceneGraph, 2007.
Available: `http://www.openscenegraph.com/`.

[75] M. Puckette. Pure Data: another integrated computer music environment. In *Proceedings, Second Intercollege Computer Music Concerts*, pages 37–41, Tachikawa, Japan, 1996.

[76] C. Ramstein and V. Hayward. The pantograph: A large workspace haptic device for a multi-modal human-computer interaction. In *CHI'94, Conference on Human Factors in Computing Systems ACM/SIGCHI Companion-4/94*, pages 57–58, Boston, MA, April 1994.

[77] Reachin Technologies. Reachin API (software).
Available: `http://www.reachin.se/products/reachinapi/`, November 2006.

[78] C. Richard and M. R. Cutkosky. Friction modeling and display in haptic applications involving user performance. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 605–611, 2002.

[79] D. Rocchesso, R. Bresin, and M. Fernström. Sounding objects. *IEEE MultiMedia*, 10 (2):42–52, 2003.

[80] J. Rovan and V. Hayward. Typology of tactile sounds and their synthesis in gesture-driven computer music performance. In M. Wanderley and M. Battier, editors, *Trends in Gestural Control of Music*, pages 297–320. IRCAM, Paris, France, 2000.

[81] K. Salisbury and C. Tarr. Haptic rendering of surfaces defined by implicit functions. In *Proceedings of the ASME Dynamic Systems and Control Division*, volume 61, pages 61–67. ASME, 1997.

[82] J. K. Salisbury Jr. Making graphics physically tangible. *Communications of the ACM*, 42(8):74–81, August 1999.

[83] SensAble Technologies. OpenHaptics Toolkit (software). Available: `http://www.sensable.com/products-openhaptics-toolkit.htm`, November 2006.

[84] SensAble Technologies. PHANTOM haptic devices (hardware). Available: `http://www.sensable.com/products-haptic-devices.htm`.

[85] S. Serafin and R. Dudas. Gestural control of a real-time physical model of a bowed string instrument. In *Proceedings of the International Computer Music Conference*, pages 375–377, Beijing, China, 1999. ICMA.

[86] S. Serafin, M. Burtner, C. Nichols, and S. O'Modhrain. Expressive controllers for bowed string physical models. In *Proceedings of the Conference on Digital Audio Effects*, pages 180–183, Limerick, Ireland, December 2001.

[87] A. Seugling and M. Rolin. Evaluation of physics engines and implementation of a physics module in a 3D-authoring tool. Master's thesis, Umeå University, March 2006.

[88] R. Smith. Open Dynamics Engine (software). Available: `http://www.ode.org`, November 2006.

[89] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3):16–20, March 2005.

[90] K. van den Doel, P. G. Kry, and D. K. Pai. FoleyAutomatic: physically-based sound effects for interactive simulation and animation. In *Proceedings of the 28th annual conference on computer graphics and interactive techniques*, pages 537–544, 2001.

[91] B. Verplank, M. Gurevich, and M. Mathews. The Plank: Designing a simple haptic controller. In *Proceedings of the Conference on New Instruments for Musical Expression*, pages 23–27, Dublin, Ireland, May 2002.

[92] W. Verplank. Haptic music exercises. In *Proceedings of the 2005 International Conference on New Interfaces for Musical Expression*, pages 256–257, Vancouver, Canada, 2005.

[93] W. Verplank and M. Mathews. Scanned synthesis. In *Proceedings of the International Computer Music Conference*, pages 368–371, Berlin, Germany, August 2000. ICMA.

[94] M. M. Wanderley and P. Depalle. Gestural control of sound synthesis. *Proceedings of the IEEE*, 92(4):632–644, April 2004. Special Issue on Engineering and Music - Supervisory Control and Auditory Communication.

[95] G. Wang and P. Cook. ChucK: a programming language for on-the-fly, real-time audio synthesis and multimedia. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 812–815, New York, NY, 2004.

[96] K. Ward, N. Galoppo, and M. C. Lin. A simulation-based VR system for interactive hairstyling. In *VR '06: Proceedings of the IEEE Virtual Reality Conference (VR 2006)*, pages 257–260, Washington, DC, 2006. IEEE Computer Society.

[97] A. Wilson, E. Larsen, D. Manocha, and M. C. Lin. Partitioning and handling massive models for interactive collision detection. In P. Brunet and R. Scopigno, editors, *Computer Graphics Forum (Eurographics '99)*, volume 18, pages 319–330. The Eurographics Association and Blackwell Publishers, 1999.

[98] M. Wright, A. Freed, and A. Momeni. OpenSound Control: State of the art 2003. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 153–159, 2003.

[99] H.-Y. Yao and V. Hayward. An experiment on length perception with a virtual rolling stone. In *Proceedings of Eurohaptics*, pages 325–330, Paris, France, July 2006.