A run-time programmable simulator to enable multi-modal interaction with rigid-body systems

Stephen Sinclair *, Marcelo M. Wanderley

Input Devices and Music Interaction Laboratory, Center for Interdisciplinary Research in Music Media and Technology, McGill University, 555 Sherbrooke St. W., Montreal, Que., Canada H3A 1E3

ABSTRACT

This paper describes DIMPLE, a software application for haptic force-feedback controllers which allows easy creation of interactive rigid-body simulations. DIMPLE makes extensive use of an established standard for control-rate transmission of audio control commands, which can be used to drive many simultaneous parameters of a given audio/visual synthesis engine.

Because it is used with a high-level, visual multimedia programming language, DIMPLE allows fast and uncomplicated development of responsive, haptically-enabled virtual environments useful for fast prototyping of applications in fields where lower level programming skills may not be widespread. Examples of specific scenes constructed using DIMPLE are given, with applications to perception, HCI research, music, and multimedia. A pilot evaluation study was performed comparing DIMPLE to another implementation of a specific scene, which showed comparable results between subjects' overall impressions of the simulation.

1. Introduction

As haptic devices become increasingly less expensive¹ and computer processing power and memory sizes continue to increase, the use of force-feedback haptics becomes commonplace in various research areas such as experimental psychology, human–computer interaction, multimedia arts and computer music.

Applications of haptic interactive virtual environments in such fields include the development of tools for user tests in cognition and action/perception experiments, the evaluation and validation of interaction metaphors and multi-modal interaction strategies in HCI, the development of artistic works in multimedia arts and the design of virtual musical instruments in computer music, to name but a few.

Many existing virtual reality (VR) applications provide excellent frameworks for the development of interactive virtual environments, often supporting rich feature sets such as a coherent and simple scene description language, textured objects, collaborative interaction, and, though more rare, support for haptic devices.

For example, XVR (Carrozzino et al., 2005) defines a Java-like scripting language which can compile to very small web applications represented as bytecode for an interpreter running on a Microsoft browser-based platform. It supports different modalities including interaction with haptic devices by synchronously running multiple simulation loops at different update rates. Haptics is supported through a project called HapticWeb² developed for use in a virtual online museum (Tecchia et al., 2007).

Many game- and multimedia-oriented scene graph APIs have been extended to include haptic support. For example, Umeå University's VRlab has created extensions for Open Scene Graph, an open-source C++ framework for 3D graphics, to allow interaction with a haptic device (Backman et al., 2007). Similarly, there have been haptic extensions created for various game engines, such as Crystal Space (Aamisepp et al., 2003) and Half-Life (Jiang et al., 2005).

In the commercial sector, several offerings are also available. Handshake proSENSE³ allows defining a virtual environment through the VRML description language, and programming it visually using Mathworks Simulink. Similarly, the Reachin' API from Reachin' Technologies uses VRML and the Python programming language to define interactive visual/haptic scenes. The open-source H3D C++ API from SenseGraphics supports a haptic scene graph using the XML-based X3D standard for 3D graphics.

¹ Recently, entry-level multi-purpose devices such as Novint Technologies' Falcon have been proposed at costs around $200 per unit.

² Unfortunately, Handshake Interactive Technologies, Inc. has recently closed its doors, so the actual availability of proSENSE is currently unknown.

0953-5438/$ - see front matter © 2008 Elsevier B.V. All rights reserved.

doi:10.1016/j.intcom.2008.10.012
Finally, recent developments at ACROE have successfully implemented interactive haptic environments at frequencies above 30 kHz using a mass-spring interaction formalism (Castet and Florens, 2008). This system utilizes special processing hardware and a strict real-time operating system to achieve synchronous concurrent processing for audio and haptics at high frequencies. This allows for an unmatched level of coherence between audio and haptic modalities because latency for the entire system is limited to the frequency requirements of the highest sensory sample rate.

Nevertheless, the development of VR and haptic applications using existing environments usually requires familiarity with programming languages such as C++, or defining the environment with an elaborate description language. While graphical modelling of VRML and X3D is possible, this rarely takes into account scene dynamics or haptic and audio feedback. With force-feedback controllers becoming pervasive in various non-engineering or -computer science related research environments, it is desirable to make available high-level software that encapsulates existing algorithms with an approachable interface, opening up the use of these devices to a broader audience of researchers, interface designers, and media artists for use in a multi-modal context.

In research areas such as experimental psychology or multimedia art, leading researchers and their students do not necessarily have formal training in programming. To make use of haptic devices in these contexts implies the learning of novel skills or the hiring of research assistants typically from computer science or engineering. Even when such skills are available, researchers may want to spend their time in experimental studies instead of in programming applications. We feel that simple simulation needs should be met by simple technical requirements. Furthermore, particularly in the case of media artists, it should be possible to leverage existing skills in media-oriented programming environments.

An important feature missing in many existing VR applications is the ability to visually and interactively program prototypes in similar ways to engineering tools. Since Puckette’s development of the Max patching system in the late 80s (Puckette, 1986), many audio and video programming environments have borrowed on the idea of allowing run-time instantiation of unit generators and real-time, visual re-organization of the connection network defining an audio/visual display. Such a dynamically programmable environment has proven to lend itself well to experimentation with media. Though it can be argued that visual programming can of course have its own learning curve, it is difficult to ignore its immense popularity with artists. To date, this kind of environment has not generally been available for the haptic modality or for VR programming as a whole.

Finally, a common limitation of most existing VR/haptic environments is their lack of support for high-quality, flexible sound generation. With a few notable exceptions (Cadoz et al., 2003), the majority of existing environments are limited to the playback of pre-recorded samples or to producing General MIDI sounds.

Flexible and powerful sound synthesis methods have been developed for more than half a century, and dozens of applications are widely used by musicians and media artists worldwide to create real-time, high-quality sounds. Meanwhile, the use of sound-file playback has been compared to the difference between the old stop-frame animation techniques and the current trends toward physically realistic motion (Raghuvanshi et al., 2007). Techniques such as waveguide modelling, modal analysis, and additive synthesis provide audio algorithms that are concretely based on empirical observations of physical systems, careful design of formats, accurate representation of acoustic spaces, and purposeful manipulation of partials to achieve specific sonic results. They are able to provide large parameter spaces which can be modified in semantically meaningful ways for true interaction with a modeled mechanical system.

Nevertheless, most VR applications, though sometimes providing advanced sound spatialization capabilities, still limit users to very basic sound generation, often exposing only timing of events to the sound engine; an understandable decision in the context of video game sound effects, but insufficient to produce truly convincing audio stimuli. We believe that sound provides important perceptual cues to the user that not only reinforce, but are essential components of the virtual experience. Since it has been extensively shown that low quality audio may affect the perception of multi-modal systems (Storms and Zyda, 2000), it may be interesting for VR designers to experiment with flexible audio synthesis software during the sound design process.

In trying to address the cited shortcomings of commonly available interactive virtual environment design tools we have developed an application called DIMPLE – Dynamically Interactive Musically Physical Environment.7 DIMPLE is based on open-source software, which combines haptic interfaces with a rigid-body simulator, and can be used with dynamically programmable Max-like media languages such as Pure Data (PD) to create touchable virtual controllers for sound or video.

This article outlines the goals of this project and describes strategies taken to overcome various limitations encountered during its development due to the choice of off-the-shelf technologies. Section 2 provides an overview of DIMPLE, while Section 3 gives details on its implementation. Section 4 provides various examples of environments developed using DIMPLE. Section 5 presents more advanced interaction techniques available with DIMPLE. A pilot study comparing a virtual implementation of the PebbleBox (O’Modhrain and Essl, 2004) in DIMPLE with a previous example (Magnusson et al., 2006) is provided in Section 6. Some conclusions and ideas for future work are presented in Section 7.

2. Overview of DIMPLE

DIMPLE is a cross-platform application designed to run on at least Linux, Microsoft Windows, and Apple’s OS X.8 It acts as a simulation “server” which can talk to audio software acting as the client. When started, the simulation contains an empty world, and sits inactive until it receives instructions.

It interoperates with other applications through the Open Sound Control protocol (Wright and Freed, 1997), a standard that has been up-and-coming as a replacement for MIDI, but is in fact convenient for any kind of message-based communication. OSC messages are tagged with a path (sometimes called the address) which fully describes the message and its arguments. It is typically carried using UDP packets, though it is formally “transport-independent”, intended for use over any physical or logical channel. It was chosen for this project primarily because it is already well-supported by the audio software we wish to communicate with, but also because its addressing scheme fits nicely with a hierarchical scene graph such as that used by DIMPLE. It supports transport of floating-point data with up to 64 bits of precision, arranged in a list format along with strings and other data types, and tagged with semantic information to fully describe the message contents, giving it a distinct advantage over MIDI’s 7 bit control values. Many users of Max-like software are already

---

4 For instance National Instruments’ LabView and Mathworks Simulink.

5 The Pure Data mailing list currently averages about 1000 messages per month. The Max/MSP list forum receives two to three times this traffic.

6 Musical Instrument Digital Interface.

7 Source code is available at http://idmil.org/software/dimple.

8 Currently, many haptic devices have drivers available only for Windows, and a few for Linux, though we are currently unaware of such a device supported on OS X. DIMPLE does run on OS X, albeit without haptic device support.
comfortable with using OSC for network communication. This makes it trivial for them to learn how to interact with DIMPLE. Lastly, OSC messages are time-stamped, making it possible to use them in a well-synchronized fashion if needed.

Messages in DIMPLE contain instructions to create or modify the behaviour of objects in the scene graph (the world). Objects consist of simple 3D shapes such as prisms and cubes. These simple shapes can be combined into more complex structures by gluing them together into compound objects. Additionally, their behaviour relative to one another can be specified by creating constraints, corresponding to different types of joints such as hinges or ball joints. Constraints may be associated with a response which determines the dynamics of the free axes, such as the coefficients of a damped spring for example.

The idea is that by combining primitives and specifying their interactive behaviour, we can use continuous information about their movement to control sound. The trajectories of connected objects can become quite complicated depending on what constraints have been specified, making interesting modulators for sonic parameters. Additionally, object–object collisions can generate events that are associated with a velocity value, and direct interaction information between the haptic device and the objects is also available. Thus, a virtual controller can be created which behaves physically and, through haptic rendering, can have embodied meaning to the user.

2.1. Virtual instruments as a gesture mapping layer

The DIMPLE virtual environment effectively becomes a complex (many to many) gesture mapping layer between the haptic device and the sound synthesis algorithm. It has been shown (e.g., Rovan et al., 1997; Hunt et al., 2003) that complex mappings are generally more satisfying for a performer than simple ones. That is, performers tend to prefer playing instruments where input parameters are cross-coupled in some way to produce complex changes in sound, rather than simply mapping, for instance, movement in one direction to timbre and movement in the other to pitch. Similarly in DIMPLE, movement of the physical device does not directly affect synthesis (unless desired), but indirectly controls the movement of modulators through physically modeled relationships. Therefore a user's single gesture can result in several simultaneous and interrelated movement trajectories through the environment's coordinate system.

Using a VR scene such as presented by DIMPLE to embody the mapping from 3D position to a parameter space can provide an explicit metaphor, easily internalized by the user. Haptic feedback further solidifies the sense of contact with a virtual object, and thus the sense of actual manipulation of a sound-producing object. Providing a sense of inertia and contour can have an important impact on skill acquisition and gestural accuracy (Rovan and Hayward, 2000). This concrete association between object movement and modulated sound can lead to an improvement in the user of “playing” the virtual objects rather than “playing” the haptic device. Mulder et al. (1998) discussed the idea of virtual musical instruments (VMI) as being a way to access the universe of sounds made possible through computational acoustics, but he acknowledged the lack of tacit feedback. DIMPLE makes haptic virtual instruments not only possible, but easy.

This is not to be confused with the approach of environments like ACROE's Genesis (Cadoz et al., 1993), or IRCAM's Modalys (Eckel et al., 1995), which mimic the mechanics of virtual acoustic models thus creating a direct and explicit link between model and sound; DIMPLE instead allows the creation of touchable virtual controllers for external and arbitrary sound synthesis engines. Any set of object parameters can be mapped through a dynamic programming environment to any set of sound parameters. Physical models can certainly be used with DIMPLE, however it is intended for controlling generalized parameter spaces, assigned by the application, rather than being tied to a specific internal model of sound and physics that must agree. In fact, we have used DIMPLE to control Modalys' real-time Max/MSP object, so it is clear that these two ideas of virtual instruments are in fact compatible, rather than in competition.

3. Implementation

DIMPLE takes advantage of several open-source libraries to implement various functionality. Haptics and graphics are taken care of by CHAI 3D (Conti et al., 2003), an open-source C++ scene graph API for haptics. CHAI calculates reflection forces using the Zilles god-object method (Zilles and Salisbury, 1995) for 3-DOF interaction. This method establishes the haptic force calculations as based on the difference in position between the actual end effector and a goal position, which cannot pass through solid surfaces. This avoids problems associated with thin objects and wall selection ambiguity which can occur when only penetration depth is considered, and is therefore appropriate for triangle mesh models. It also has routines to provide OpenGL-based rendering. A nice feature of CHAI is that it supports several commercial haptic devices, detected at run-time, making it possible to use DIMPLE with any of these without the need to recompile. Rigid-body dynamics are implemented using the Open Dynamics Engine (Smith, 2006).

One challenge we encountered in establishing a link between these libraries is that each of them requires its own independent set of data structures to maintain the scene. Sometimes even basic representations may differ; for instance, while CHAI uses a mesh representation of the prism objects, it is more efficient for ODE to make use of the box object type, a solid geometry model of a rectangular prism. Thus it was necessary to implement a class structure which could encompass these differences and provide a common Open Sound Control layer to access object properties.

Additionally, the plan was to allow each simulation to run in parallel in order to take advantage of multi-core processors, meaning that synchronization and other concurrency issues were of concern. Since multiple copies of the scene had to be kept synchronized regardless of concurrency, we found that internal messaging between the independent scene graphs filled both roles nicely.

3.1. Concurrency model

The DIMPLE internals are organized around a C++ class framework that abstracts a tree of OSC-addressable objects. Every level of the scene hierarchy has properties that can be addressed and requested via OSC messaging. The simulation is divided into several concurrent models, and each of these, running in parallel using POSIX threads, is represented by a subclass of the lowest-level Simulation class. Simulation represents the “world”, or global object manager, for each type of simulation and contains the list of objects and constraints in the world as well as a list of other simulations that it should communicate with when something changes.

There is a PhysicsSim subclass and a HapticsSim subclass for the haptic device, a VisualSim subclass for the OpenGL display, and an InterfaceSim subclass which handles incoming client messages to forward to the appropriate destination. For example, a PhysicsSim object sends a message to the other simulations when it moves, and a HapticsSim object sends messages to the corresponding physics object when force is applied to it. Equivalently, a client can send a force message to an object which will

---

5 This idea was borrowed from a CHAI demonstration applet.
forward to the PhysicsSim, or to the haptic cursor, which go to the HapticsSim. Some messages, like requesting the creation of an object, will go to all simulations. From the client’s point of view, it is acting on a single coherent simulation.

Since they are independent, there is no requirement that the simulations should run at the same rate or even in evenly divisible intervals of each other. The default configuration is to have the physics simulation at 100 Hz while the haptic loop executes at 1 kHz and graphics at 30 Hz. Fig. 1 shows the internal information flow within DIMPLE.

3.2. Communication

This separation between concurrent simulations implies that all communication must be performed over a thread-safe messaging system. Additionally, each simulation must execute under real-time constraints, so a non-blocking solution is desirable. Since scene objects are already designed to respond to OSC messages for communication with the client software, we opted to use the same mechanism for inter-simulation synchronization. Thus, OSC is not only used for communication with the client process, but also serves as an internal protocol.

An argument here is that OSC can be verbose and require some processing overhead for parsing the path identifier. In practice, we have found that this overhead is mostly negligible in comparison to calculations for collision detection, rigid-body dynamics, and force-feedback computation. On the other hand, time spent in system calls can be detrimental: this includes accessing the network API as well as performing memory allocation/de-allocation, factors that are well-known in real-time programming theory. To improve this, we have replaced the usual UDP-based transport with a wait-free circular queue for passing OSC data between threads, similar to the technique described in (Chen and Burns, 1998). We are thus able to retain the advantages of OSC (extensibility, interoperability, ease of use) while speeding up communication on the local host. Fall-back to standard network communication if distributed operation is desired is also made possible.

One impact of this is that it is trivial to change the concurrency topology. For instance, the physics simulation can be made to run within the haptics thread, where they pass OSC messages to each other with zero delay, allowing the more usual synchronous interaction between haptics and physics processes.

3.3. Class organization

A basic set of classes, called OscBase, implementing OSC-addressable objects wraps the LibLo (Harris et al., 2005) library. Property objects (OscValue) are used to create addressable properties that can be set and retrieved over OSC. The Simulation class, which extends OscBase, contains factory classes (ShapeFactory) for each type of supported scene object. All ShapeFactory instances generate OscObject instances on demand, but specific ShapeFactory subclasses generate specific OscObject subclasses. Thus, the physics simulation’s factories will generate ODE box and sphere “geoms”, each with a “body” data structure for dynamics (see the ODE manual [Smith, 2006]), while the haptics simulation’s factories will generate CHAI objects of type cMesh or cShapeSphere.10 Once created, each shape subclass has properties (OscVector3, OscScalar, or OscString instances), appropriate to its subclass, which can be modified and retrieved using messages. So all OscSphere objects have a “radius” property, but only OscSphereCHAI has a “stiffness” property. These messages are responded to appropriately for each simulation type through a set of callback functions defined by the specific subclass. Some of this class structure is shown in Fig. 2.

---

10 cShapeSphere is one of CHAI’s “implicit surfaces”, which defines a smooth sphere rather than being composed of triangular sections.
3.4. Message namespace

Since each object is associated with a name, the OSC namespace defined in DIMPLE changes dynamically as objects are created and destroyed. In other words, the only accessible messages when DIMPLE is first initialized are to create objects (/dimple/sphere/create, /dimple/prism/create). These create messages are received by the ShapeFactory instances, which generate an appropriate object. On creation, objects register their own sub-addresses within the DIMPLE namespace. So, properties and methods for an object called test can be accessed with the /dimple/test/ prefix.

This means that the calling application has the job of keeping track of created objects. This can be done in PD quite easily by using numbered strings to name multiple similar objects. Thanks to OSC’s pattern matching specification (Wright and Freed, 1997), it is also possible to send messages to several objects at once, which can be useful if named objects follow a pattern. For example, all spheres, which we have named sp1, sp2, sp3, etc., can be sent flying in the Z direction with the message, /dimple/sp*/force 0 0 10.

To extract useful streams of information from object movement, the application can request that any given property be announced at regularly timed intervals using the /get suffix. Collisions can also be requested for any specific object (/collide), or between any two objects in the scene. The responding message gives information about which two objects collided and at what combined velocity.

4. Examples

In this section, we will illustrate the ease of creating a virtual instrument in DIMPLE by giving a short example of how to construct a simple interaction. The subsequent sections will briefly describe some examples of previous applications and potential uses for DIMPLE.

4.1. Coupled oscillation with the hinge constraint

We shall create a simple PD patch controlling two spheres connected by a hinge joint, where the movement of the second sphere will modulate the frequency of a sinusoid.

We begin by setting up communication between DIMPLE and PD. This can be done by instantiating the sendOSC and dumpOSC objects, and routing them using send and receive objects (send and receive), as in Fig. 3a. Three messages are used to create a sphere in a given location, and to set its colour and radius. Two spheres are created this way by sending two sets of parameters to these message objects, as in Fig. 3b. Next, two hinges are created, one between the blue sphere and the red sphere centered on the red sphere’s location, and the other between the red sphere and the “world”, centered at the origin, as in Fig. 3c. The hinge creation message takes six values: a 3-vector for the position and another 3-vector for the axis. The position of the blue sphere is requested at 10-ms intervals. Finally, the Z-axis value of the incoming position messages is scaled and used to set the frequency of a sinusoid, as in Fig. 3d. To hear the frequency change, the user can interact with the spheres using a haptic device, or send either sphere a /force message which would cause coupled oscillation of both objects.

4.2. MarbleBox

One of the initial test programs created using DIMPLE was a virtual implementation of the PebbleBox controller (O’Modhrain and
Essl, 2004). The simple idea behind this physical gestural controller is to use a microphone to detect collisions between many small pebbles and trigger a granular synthesis routine to create new sounds based on the interaction. A virtual-haptic version of this controller has in fact been implemented previously in two instances (Magnusson et al., 2006). One of these, which we’ll refer to herein as the Virtual PebbleBox (VPB), was an application based on SensAble’s OpenHaptics for use with a Phantom controller, in combination with the Open Dynamics Engine for rigid-body physics. The other used the TGR haptic device from ERGOS technologies and was executed on a dedicated DSP analog I/O board. In the latter implementation, the sound was produced by a mass-spring model sampled at 30 kHz, while haptics was rendered through the same model decimated to 3 kHz. In the OpenHaptics implementation, sound was produced by triggering a sound-file on collision events calculated at 1 kHz. Both models used spheres to replace the pebbles.

This fairly simple interactive idea provided a nice case study for comparison with DIMPLE. A PD patch for DIMPLE, shown in Fig. 4, was developed to present a similar scenario, with spheres rolling around inside a box, and sound created on collisions by sending an impulse into a simple modal synthesizer implemented with a few biquad filters (Sinclair and Wanderley, 2007). The “MarbleBox” patch consists of a few messages to create, size and constrain the 5-sided box, and to create a sphere and set its mass and radius. One more message is needed to request collision information. The sphere-creation routine can be triggered as many times as one likes, filling the box with marbles.

That a small patch expressed in a visual language can quickly recreate the work of a dedicated C++ application shows that a generalized interface like DIMPLE has some use in experimentation with virtual controllers. Additionally it was trivial to modify the patch to trigger the sound on another computer on the network which was running a more advanced sound synthesis routine. A further minor modification, without stopping the application, caused the pitch of the sound to change depending on which two marbles collided. The same application has been executed with a SensAble Phantom Omni, an MPBT Freedom 6S controller, and a Novint Falcon, without any code modifications.

Because a similar implementation of a Virtual PebbleBox was independently developed, this example provided a convenient scenario for a comparison study. This is described in more detail in Section 6.

---

**Fig. 3.** Portions of the example Pure Data patch demonstrating creation of spheres on hinges used to modulate a sinusoid. These dataflow diagrams, which are screenshots of the actual Pure Data “code”, are described in Section 4.1.

**Fig. 4.** (a) A Pure Data patch which constructs a controller consisting of several marbles in a box which can be pushed around to create sound events. (b) DIMPLE’s visual representation of the haptic proxy interacting with the MarbleBox.
4.3. Control of spatialization

In some instances we have used DIMPLE with external controllers which were not force-feedback devices (Marshall et al., 2007). For instance, we have used Max/MSP to connect a drum trigger to DIMPLE, creating spheres and launching them toward the horizon in various directions, the launch power determined by the hit velocity. The sound of the drum hit was recorded and looped, while the trajectory of the “cannon balls” determined the spatial positioning of the sound source in a multi-speaker display. Widely spaced and strong drum hits would send the spheres flying away from the performer. Quicker drum hits would create a series of spheres in succession, which would inevitably hit each other and fly in unpredictable directions. Lighter hits resulted in spheres rolling more slowly once they hit the virtual floor, giving the performer a degree of control over the speed of the source movement.

In another instance, we used a sensing floor to determine the horizontal projection of a user’s center of mass. DIMPLE contained several spheres within a box, similar to the MarbleBox configuration. When the user leaned in one direction or another, DIMPLE’s gravity vector would change accordingly, causing the spheres to rush towards different sides of the box, colliding in unpredictable ways. The positions of each sphere were used again to position a group of sound sources, this time produced by a granular synthesizer.

4.4. Use in HCI and perception tasks

Many experimental procedures in haptic perception, learning, and usability require simple interactive scenarios. For instance, a pilot experiment conducted in the music technology area at McGill involved a virtual plate with variable stiffness to be struck at repeated intervals using a SensAble Phantom controller (Giordano et al., 2007). Subjects were played an accompanying metallic sound of variable stiffness not necessarily correlated with the haptic stimuli. The interactive aspects of this session could have been constructed in DIMPLE with quite a trivial PD patch. Since DIMPLE was not yet available at the time the study was designed, the haptic software was commissioned to be written in C++ by an outside party.

As another example, an on-going study in collaboration with McGill’s Multi-modal Interaction Lab will use DIMPLE to present textured surfaces to subjects, with the texture parameters gradually changing towards a target location, while an audio cue also changes correspondingly. DIMPLE can report the proximate location, so that the program can react when the user reaches the target. With DIMPLE doing the hard work of displaying the textured surface (Section 5.1), an external script was made to generate the texture’s height map and this is triggered by the same PD patch which controls the audio.

This style of programming is generally much more accessible and convenient for researchers, allowing easy visual modification of program flow during execution without the need for off-line compilation and linking steps, which shortens the run-debug cycle to a few mouse clicks. Since information about object and proxy movement can be accessed on demand, user interaction can be recorded completely for later analysis and playback.

5. Other interaction techniques

The interactions described thus far allow modulation of sound based purely on object movement. This effectively provides only one way of interacting with the environment: pushing. We recognize that many other actions such as scratching, striking, plucking, bowing and damping are important in various interaction contexts. Such actions are important for achieving the subtlety that is required in musical performance, as well as other potential areas such as HCI, and cannot be ignored by a virtual environment purporting to provide a sonically interactive experience. Though requirements for these interaction techniques remains a subject of on-going research, DIMPLE provides several features which can be used to incorporate some of these more engaging actions (Sinclair and Wanderley, 2007).

5.1. Scratching textures

A relatively early use of force-feedback devices was to explore the rendering of virtual textures (Minsky et al., 1990). Indeed the concept of modulating the shape or normal of an otherwise flat surface by some procedural function or stored bitmap is a trick that borrows to a large degree from well-known techniques in graphical 3D rendering. Yet, as we see in graphics, this simple modulation can provide a greater degree of realism to the appearance of a virtual object, to the point where today it is almost always used where processing power allows. In haptic interaction too, we can provide a more realistic experience by having surfaces which are not entirely flat (Fukui, 1996). Indeed, processing power is less of a concern for haptic textures, since only points in the immediate vicinity of the end effector location need be considered. We have found that the interaction between the haptic proxy point and the surface actually provides rather interesting data for sonification.

DIMPLE allows the user to specify a bitmap which is loaded onto the surface of an object in the scene. This bitmap is displayed graphically and also haptically rendered using a force-shaded height mapping routine. As the user interacts with the surface of the object, continuous information about lateral force impulses can be directed to the audio synthesis algorithm. If passed through a modal synthesis algorithm, for example, the sound of continually rubbing a drumhead can be achieved. An example of recorded texture interaction data can be seen in Fig. 5.

One problem with this technique is that the sample rate of the haptic device is often around 1000 Hz, which is on the one hand far below the sample rate of audio, meaning that some information that would be present in the real interaction is lost, and on the other hand quite fast for sending continuous data across a network connection in short packets. DIMPLE handles this latter problem by packing several samples together into a single message which can be sent at longer intervals, or transmitted as an audio signal as discussed in the next section. Since the JND of audio-haptic latency falls somewhere below 24 ms (Adelstein et al., 2003), several samples at this rate can be safely buffered without perceivable effects. The former poses a problem which requires either re-synthesis of missing pulses, or an elevation of the sample rate.

5.2. Signal feedback

It was originally envisioned, as we have described here, that all communication with DIMPLE would occur over OSC, and consist of control messages. However, in acoustic instruments it is common that one can feel a vibrational response when the instrument’s resonating body is stimulated. This vibration is commonly thought to contribute to a player’s perception of the instrument being “alive” and responsive (Birnbaum, 2007). Though DIMPLE makes no attempt to model the actual acoustics of vibrating bodies, it does provide a method for sound synthesis models to feed generated audio signals back into DIMPLE as an audio signal. DIMPLE then samples this down to the haptic rate and feeds it into the end effector, either directly, or modulated by the haptic rendering algorithm’s force vector. This latter method gives the perception of
for playing back a recorded bell sound-file in response to object touching a vibrating body, since vibrations are felt in proportion to how hard the user pushes on an object. Of course, the control patch could then detect that the user is touching the object and apply damping to the audio signal to complete the feedback cycle.

This functionality is currently only available through a special build of DIMPLE as a PD and Max/MSP object (Sinclair and Wanderley, 2007). The object, named dimple~ in correspondence with PD naming convention, provides a variable number of signal inputs which can be connected to any audio algorithm. Signal vectors, which are transmitted during PD’s DSP callback routine, are passed through a resampling library (Castro and libsamplerate (software), 2002), which provides several down-sampling algorithms including one based on Smith and Gossett (Smith and Gossett, 1984), and then buffered for transmission to the haptic routine. This same object can provide parameter output streams at up-sampled audio rate through dimple~’s signal outlets.

6. Evaluation

We wished to investigate how subjective evaluation of the simulation’s overall “naturalness” would degrade when compared to other software, due to the use of asynchronous communication between threads with different frequencies.

As stated in Section 4.2, we decided to compare DIMPLE’s MarbleBox example against Magnusson’s Virtual PebbleBox (VPB) implementation, which used equivalent technology to achieve a similar multi-modal scenario. In a pilot study, we have modified the VPB source code to make it visually resemble (though not perfectly) the DIMPLE implementation.

While the two programs used similar techniques, some differences between the two simulations are the following: VPB executed the ODE timestep in synchrony with haptic rendering at 1 kHz and represented the haptic cursor by an ODE mass attached to the device position by a spring (a so-called “virtual coupling” technique (Colgate et al., 1995)); DIMPLE was running the physics engine asynchronously at 100 Hz, with haptic interaction happening at a zero-radius point (represented by a sphere) updated at 1 kHz. For audio feedback, VPB used a 3D-positioned sound source for playing back a recorded bell sound-file in response to object collisions. 3D positioning was implemented through the Microsoft DirectSound API. Volume was not scaled according to impact velocity, and each object was associated with its own pre-recorded sound-file of various pitches. DIMPLE (through Pure Data) issued audio feedback by sending a short impulse, scaled according to impact velocity, through a 6-mode resonator model implemented using biquads. Each object was randomly associated with one of four possible pitches, and the resulting sound was played the same on both ears of the headphones (i.e., mono).

We invited 13 participants to try both demonstrations in succession. Order of presentation was counterbalanced. All participants were between 25 and 35 years of age, where 10 participants were music or music technology graduate students at McGill University. A SensAble Phantom Omni was used for this test, and all but one subject had little to no experience using a force-feedback device.

A pen-and-paper questionnaire was used to get Likert ratings on a 7-point scale for six questions, the means and standard deviations of which can be seen in Fig. 6. We asked the participants: one question on the solidity of static objects (Flat); two questions on physical dynamics, (ease of manipulation Push and “responsiveness” Resp); two questions on sound, (“realism," Sound, and whether it matches the scenario, Match); and one question on the overall “naturalness” of the simulation (Natur). For all questions, a higher rating was better. After the task, free-form comments comparing the two simulations were also obtained.

It should be noted that seven of the participants had previously at least seen a screenshot of DIMPLE and were able to recognize it during the study. We therefore divided participants into two groups (those who knew DIMPLE and the others) and performed unpaired t-tests to determine whether this had a significant effect. We found a significant difference between the two groups only for the question on the responsiveness of the DIMPLE simulation (Resp: t = 2.95, df = 5.83, p < 0.05), in which DIMPLE performed worse. (The Sound parameter for DIMPLE and the Flat parameter for VPB had differences in standard deviation between the two groups greater than 50%—therefore, in some cases we cannot draw conclusions on this point.) No significant differences were found between groups based on presentation order.

Fig. 5. Recording of lateral impulses from dragging the haptic proxy across a texture. (a) Impulses spaced at 1 ms. (b) Linear interpolation to audio rate. (c) Driving low-frequency modal resonators.
Paired \( t \)-tests were used to determine significance of the results. These showed no significant differences for the solidity of flat surfaces (Flat: \( t = -0.54, df = 12, p = 0.60 \)), for naturalness of pushing objects (Push: \( t = -1.29, df = 12, p = 0.22 \)), or for overall naturalness (Natur: \( t = 0.18, df = 11, p = 0.86 \)). The responsiveness was judged significantly better in VPB (Resp: \( t = -2.36, df = 12, p < 0.05 \)), which was expected due to the different sample rates. Several users noted the difference between using a rigid-body sphere with a radius versus DIMPLE's point-like cursor, and seemed to prefer the former.

The questions on sound were both significant (Sound: \( t = 3.00, df = 12, p < 0.05 \), Match: \( t = 2.92, p < 0.05 \)) with a preference for DIMPLE's PD-driven modal synthesis. Free-form comments showed that users noticed many different types of cues here: some noticed the 3D positioning in VPB, others noticed the variety of pitches, and still others noticed the relationship between collision velocity and sound amplitude. It is clear that many of these could be improved in VPB with a little work on the audio engine, so we do not mean to claim any inherent perceptual properties of sound-file-based audio. However, it was interesting to note how the quality of sound seemed to have an effect on the total simulation experience. The more well-modulated sound (using velocity information, and with audible differences between instances due to resonance) was preferred.

More importantly, we found that most users, despite noticing the disparity, did not consider the degraded physics in DIMPLE to be a significant factor in their overall rating of the simulation. This is interesting because the authors have found informally that the difference in update rate for the physics simulation is clearly perceptible, and does go away when DIMPLE's physics engine is run at higher speeds. Most participants who noticed a difference expressed it as a change in “weight”, though the two simulations also were only roughly calibrated to have objects of the same mass, so further work would be needed to judge the influence of update rate on perception of weight. Nonetheless, we find it interesting to consider that while high speeds are required for haptic stability and stiff surfaces, it may be possible to degrade the speed of physical dynamics gradually when needed without harshly disturbing the user experience. This could perhaps be considered analogous to lowering the pixel density or frame rate of a visual display.

7. Conclusion

In this paper, we have presented DIMPLE, an application which allows researchers to create haptic virtual controllers for sound synthesis using any dynamic, high-level language supporting the Open Sound Control protocol.

Several examples of applications made using DIMPLE were given, showing the flexibility of the approach taken. Although many of the examples presented have fairly simple scenes and sound capabilities, OSC-compatible audio engines like Pure Data, Max/MSP, Modalys, SuperCollider, or ChucK, can easily bring to DIMPLE the power of cutting-edge sound synthesis. Conversely, consolidating several known haptic rendering techniques into an accessible interface makes it possible for non-programmers to take advantage of haptic devices.

We hope that enabling fast and easy experimentation in the design of new controllers may lead to discoveries within sonic spaces that otherwise might have taken much more time and effort to achieve (Cook, 2001). DIMPLE in this context should prove useful as a rapid prototyping tool for gestural controllers in many research areas.

Limitations implied by the DIMPLE approach of encapsulating specific physical modelling routines for off-the-shelf devices include a reduced simulation rate as compared to the state-of-the-art (Luciani et al., 2007). Some applications may indeed require much higher simulation rates or more flexible control of real-time rendering algorithms. In these cases, more advanced solutions or different hardware may be required. But even in such extreme cases, DIMPLE may prove useful for prototyping preliminary, proof-of-concept versions of such applications before full development in another environment is required. With a little extra work, it is also possible to add new objects and algorithms to DIMPLE, or to port parts of it to high-performance systems. For example, in at least one instance DIMPLE has been extended to integrate a different physical modelling approach (Erkut et al., 2008).

We plan to further study the relationship between haptic and audio quality, to determine more exactly how these interact to balance the feeling of multi-modal immersion. This may include permutations of the study presented here involving different sounds, with and without visuals, and at varying haptic update frequencies. Such studies will hopefully shed light on how different levels of quality in haptic feedback affect actual performance.

Acknowledgements

This work was sponsored in part by an Industrial Postgraduate Scholarship from the Natural Sciences and Engineering Research Council of Canada (NSERC) to the first author and by grants from NSERC (Discovery and Special Research Opportunity), the Canadian...
Foundation for Innovation (CFI) to the second author, as well as to the Enactive Network 6th Framework European Project. We are very grateful to Charlotte Magnusson for providing us with the source code for the VPB. Thanks also to Joseph Malloch and Mark Marshall for experimenting with DIMPLE for gestural control of spatialization and to Mark for help with the experiment. Finally, thanks to the anonymous reviewers for detailed comments and suggestions.

References


